

Motion Planning for Multi-Link Robots with Artificial Potential Fields and Modified Simulated Annealing

By

Deval Yagnik

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science

In

The Faculty of Engineering and Applied Science

Electrical and Computer Engineering Program

University of Ontario Institute of Technology
December, 2010
© Deval Yagnik, 2010

CERTIFICATE OF APPROVAL

Submitted by: DEVAL YAGNIK Student #: 100388416
First Name, Last Name

In partial fulfillment of the requirements for the degree of:

Master of Applied Science in Electrical & Computer Engineering
Degree Name in full Name of Program

Date of Defence (if applicable): 3rd December 2010

Thesis Title:

Motion Planning of Multi-Link Robots With Artificial Potential Field and Modified Simulated Annealing

The undersigned certify that they recommend this thesis to the Office of Graduate Studies for acceptance:

Chair of Examining Committee	Signature	Date (yyyy/mm/dd)
------------------------------	-----------	-------------------

External Examiner	Signature	Date (yyyy/mm/dd)
-------------------	-----------	-------------------

Member of Examining Committee	Signature	Date (yyyy/mm/dd)
-------------------------------	-----------	-------------------

Member of Examining Committee	Signature	Date (yyyy/mm/dd)
-------------------------------	-----------	-------------------

As research supervisor for the above student, I certify that I have read the following defended thesis, have approved changes required by the final examiners, and recommend it to the Office of Graduate Studies for acceptance:

Name of Research Supervisor	Signature of Research Supervisor	Date (yyyy/mm/dd)
-----------------------------	----------------------------------	-------------------

Abstract

In this thesis we present a hybrid control methodology using Artificial Potential Fields (APF) integrated with a modified Simulated Annealing (SA) optimization algorithm for motion planning of a multi-link robots team. The principle of this work is based on the locomotion of a snake where subsequent links follow the trace of the head. The proposed algorithm uses the APF method which provides simple, efficient and effective path planning and the modified SA is applied in order for the robots to recover from a local minima. Modifications to the SA algorithm improve the performance of the algorithm and reduce convergence time.

Validation on a three-link snake robot shows that the derived control laws from the motion planning algorithm that combine APF and SA can successfully navigate the robot to reach its destination, while avoiding collisions with multiple obstacles and other robots in its path as well as recover from local minima. To improve the performance of the algorithm, the gradient descent method is replaced by Newton's method which helps in reducing the zigzagging phenomenon in gradient descent method while the robot moves in the vicinity of an obstacle.

Keywords—Artificial Potential fields, Motion Planning, Multiple Link Robot, Simulated Annealing, Snake Robot

ACKNOWLEDGEMENTS

First of all, I would like to thank and express my gratitude to my supervisors, Dr. Jing Ren and Dr. Ramiro Liscano for their expertise, support and constructive guidance throughout my research work at UOIT. It was Dr. Ren's confidence in me that encouraged me to pursue my studies as a MASc student, and I am indebted to Dr. Liscano for his priceless assistance. It has been an honor working with them both.

I would also like to thank, and to praise, the faculty, staff and my fellow graduate students for the guidance and help throughout my masters program. Special thanks to Dr. Ying Wang for her care and attention, and for giving me the opportunity of working as a teaching assistant for the first time. I want to thank Dr. Lixuan Lu for being a part of my advisory committee by sparing precious time from her schedule.

Above all, I thank my family and friends who consistently stood by my side, encouraging me no matter the day or time.

Table of Contents

Certificate of Approval.....	I
Abstract.....	II
Acknowledgement.....	III
Table of Contents.....	IV
List of Figures.....	IX
List of Tables and Codes.....	XI
List of Acronyms and Variables.....	XII
1. Introduction.....	1
1.1 Motion Planning Problem.....	1
1.1.1 Different Tasks or Applications of Motion Planning.....	3
1.1.2 Benefits of Multi-Link Robot Teams.....	5
1.1.3 Artificial Potential Field Method to Solve Motion Planning Problem.....	6
1.2 Problem Statement.....	7
1.3 Thesis Contribution.....	7
1.3.1 Multi-Link Motion Planning.....	7
1.3.2 Modified Simulated Annealing for Local Minima Recovery.....	8
1.3.3 IEEE/ASME Conference Paper.....	8
1.4 Outline of Chapters.....	9
2. Background and Related Work.....	10
2.1 Different Techniques to Solve Motion Planning Problem.....	10
2.1.1 Implicit and Explicit Motion Planning.....	10

2.1.2	Intelligent Techniques and Control Theory Based Techniques.....	11
2.1.3	Artificial Potential Field Method.....	15
2.2	Multi-Link Motion Planning With APF.....	16
2.2.1	Head Follower Tails with APF.....	17
2.2.2	Gradient Descent and Newton's Method.....	19
2.2.3	Limitations of APF Method.....	19
2.3	Local Minima Recovery.....	20
2.4	Proposed Approach to Local Minima Problem.....	21
2.5	Assumptions of Robot Capabilities.....	22
3.	Artificial Potential Field (APF) Method and Gaussian Functions.....	23
3.1	Artificial Potential Field.....	23
3.1.1	Gaussian Models.....	23
3.1.1.1	Gaussian Functions for Attractors.....	24
3.1.1.2	Gaussian Functions for Obstacles.....	25
3.1.1.3	Model of Work Space.....	27
3.1.2	Navigation Function.....	29
3.1.3	Control Law.....	30
3.2	Limitations of APF.....	31
3.2.1	Local Minima.....	32
3.2.2	GNRON Problem.....	34
3.2.3	No Passage Between Closely Spaced Obstacles.....	36
3.2.4	Oscillations in the Presence of Obstacles and Narrow Passages.....	36
4.	Multi-Link Robot Navigation with APF.....	37

4.1 Head Navigation.....	37
4.1.1 Gradient Descent Method.....	37
4.1.2 Newton's Method.....	38
4.2 Tail Navigation.....	40
4.2.1 Inner Product and Direction of Tail.....	41
4.3 Activity Diagrams for the Multi-Link Robot Navigation.....	43
4.3.1 Algorithmic Motion Planning.....	43
4.3.2 Conditions to be Satisfied.....	45
4.3.3 Movement During APF Domain.....	45
5. Local Minima Recovery.....	47
5.1 Simulated Annealing for Local Minima Recovery.....	47
5.1.1 Simulated Annealing.....	47
5.1.2 Parameters for Annealing Schedule.....	48
5.1.3 Acceptance Probability.....	49
5.2 Modifications in SA.....	50
5.2.1 Comparison Between Two Approaches.....	52
5.3 Integration of APF and SA to Recover From Local Minima.....	53
5.4 Activity Diagram of Recovery from Local Minima with Help of Modified SA and APF Integration.....	56
6. Implementation and Simulation Results.....	58
6.1 Implementation of Algorithm.....	58
6.1.1 Use Case Diagram.....	58
6.1.2 Message Diagrams.....	59

6.1.2.1 Overview of Motion Planner.....	59
6.1.2.2 Overview of the Algorithm.....	60
6.1.2.3 Overview of Artificial Potential Field Method For Motion Planning.....	61
6.1.2.4 Local Minima Recovery.....	62
6.2 Simulation Scenarios.....	64
6.2.1 Performance of the Algorithm in Between Closely Spaced Obstacles.....	64
6.2.2 Local Minima Recovery.....	66
6.2.3 Multiple Robots – Cooperation.....	69
6.3 Rescue Work Scenario.....	71
7. Conclusion and Future Work.....	73
7.1 Conclusion.....	73
7.2 Possible Future Work.....	74
REFERENCES.....	75
APPENDIX: MATLAB Simulation Code.....	82

List of Figures

Figure 1: Motion Planning Problem.....	1
Figure 2: Objective of Motion Planning Algorithm.....	2
Figure 3: Multi-Link Robot Navigation.....	18
Figure 4(a-b): Contour Plots of Gaussian Function for Attractor Over the Work Space in 2-D and 3-D.....	25
Figure 5(a-b): Contour Plots of Gaussian Function for Obstacles Over the Work Space in 2-D and 3-D.....	27
Figure 6: Model of Workspace with a Single Target and Four Obstacles Generated with a Gaussian Function.....	28
Figure 7: Contour Plots for the Goal and 4 Obstacles in the Workspace.....	29
Figure 8: Vector Diagrams of Velocity Enforced by Attractor, by Obstacle and the Resultant Velocity.....	31
Figure 9: Gaussian Function Avoids Some of the Undesired Local Minima.....	32
Figure 10: Local Minimum in Multi Obstacle Workspace.....	33
Figure 11: GNRON Problem in APF.....	35
Figure 12: GNRON Problem Solution with Gaussian Function.....	35
Figure 13: Comparison between Gradient Descent and Newton's Method.....	39
Figure 14: Comparison between Gradient Descent and Newton's Method for Multi-Link Robot.....	39
Figure 15: Diagram Shows the Heading Direction of the Link (d_h), Perpendicular Direction (\dot{d}_h) to the Heading Direction, the Negative Gradient Direction (d_{ng}) of the Tail.....	41

Figure 16: Tail Movement in Accordance with the Inner Product.....	43
Figure 17: Activity Diagram 1.....	44
Figure 18: Activity Diagram 2.....	45
Figure 19: Activity Diagram 3.....	46
Figure 20: Relation between Number of Iteration t and Decay of Temperature.....	48
Figure 21: Relation between Number of Iteration t and Acceptance Probability $P(t)$	50
Figure 22: Local Minima Recovery with Modified SA.....	52
Figure 23: Activity Diagram 4.....	57
Figure 24: Use Case Diagram of Motion Planner.....	58
Figure 25: Message Diagram: Overview of Motion Planning System.....	59
Figure 26: Message Diagram: Algorithm Overview.....	61
Figure 27: Message Diagram: APF Method.....	62
Figure 28: Message Diagram: Local Minima Recovery.....	63
Figure 29: Simulation for 3 Link Robot in Narrow Passage.....	66
Figure 30: Local Minima Trap without SA.....	67
Figure 31: Local Minima Recovery with Modified SA.....	68
Figure 32: Comparison between SA and Modified SA Performance.....	69
Figure 33: Co-ordination between Two Multi-Link Robots.....	71
Figure 34: Rescue Operation Scenarios.....	72

List of Tables and Codes

Table 1: Comparison between SA and Modified SA.....	53
Code 1: Local Minima Recovery.....	55

List of Acronyms and Variables

APF: Artificial Potential Field

SA: Simulated Annealing

GNRON: Goal Non-Reachable with Obstacle Nearby

GPS: Global Positioning System

σ : Variance of Gaussian function

C: Positive Integer to Specify Range of Obstacle

(x,y): Coordinates in x and y axes

α : Gain Factor for Speed of Robot

d_h : Heading Direction of the Link

\dot{d}_h : Direction Perpendicular to the Heading Direction

d_{ng} : Direction of the Negative Gradient for Tail

β : Angle between d_h and d_{ng}

θ : Angle between Heading Direction and x-axis

φ : Angle between d_{ng} and \dot{d}_h

I_p : Inner Product in between d_h and d_{ng}

T: Temperature Parameter for Annealing Schedule

α_t : Fraction by Which the Temperature Parameter Decreases

T0: Initial Temperature

P(t): Acceptance Probability at every step t

r : Randomly Derived Integer for Acceptance Probability

CHAPTER 1

INTRODUCTION

1.1 Motion Planning Problem

Robotics is a branch of engineering which involves a variety of areas such as design, locomotion, manufacturing, motion planning etc. Motion planning is the method of detailing or describing a task into atomic motions. It has several applications in robotics, including autonomy, automation, and robot design, as well as applications in other fields, such as animating digital characters, architectural design, robotic surgery, and the study of biological molecules.

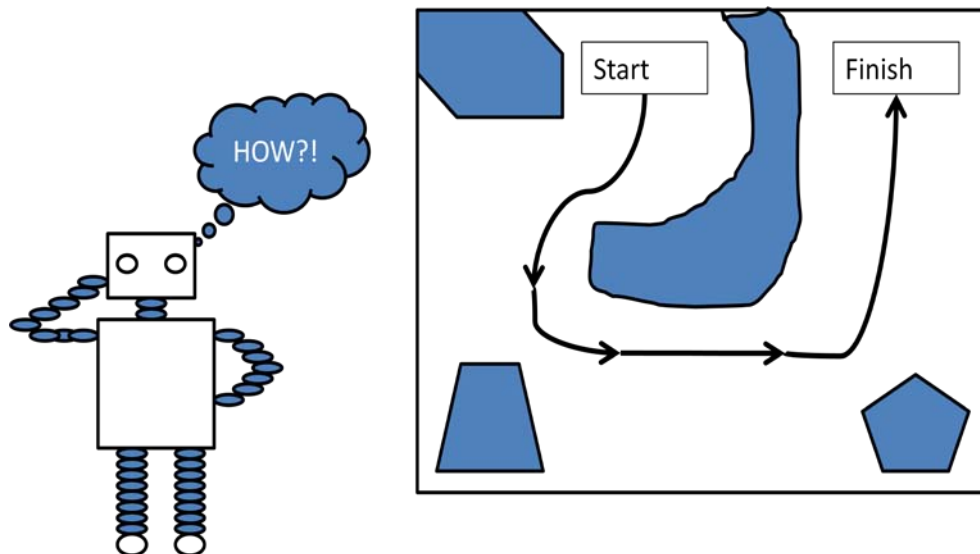


Figure 1 Motion Planning Problem

In mobile robot applications, motion planning algorithms navigate a mobile robot inside a building to a distant waypoint. While doing so it should make sure that the robot avoids obstacles and successfully reaches the objective as shown in Figure 1. A motion planning algorithm basically takes input in the form of a task description and knowledge of work space. As an output it gives the speed and orientation commands for the robot movement. Figure 2 describes a simple scenario of motion planning implementation. These commands are for the controls mounted on the robot. In robotics motion planning, algorithms can be used for robots with a larger number of joints or degrees of freedom. For example, manipulators, or arm robots in industries, are able to perform more complex tasks like manipulation of objects, have different constraints, and work in uncertain environments, or with an imperfection in the robot-environment model.

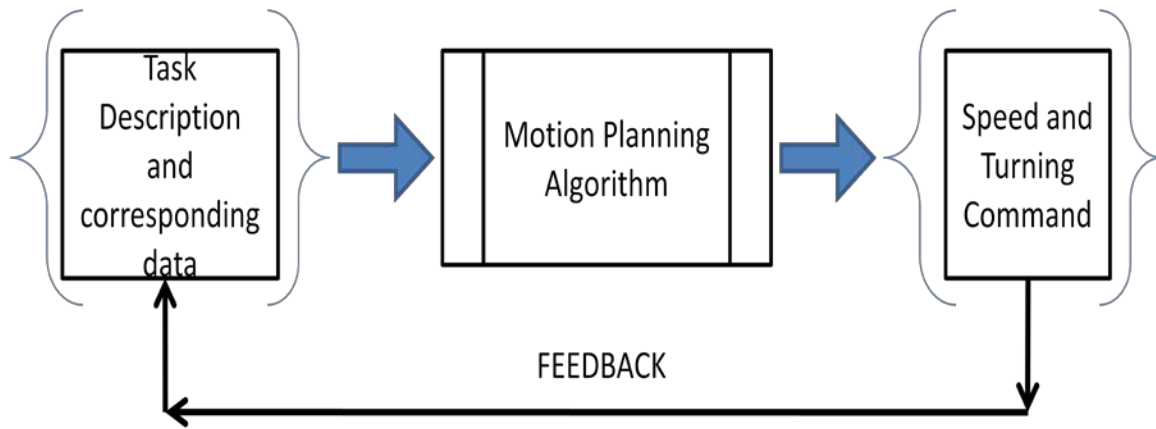


Figure 2 Objective of Motion Planning Algorithm

Conclusively, we can describe a basic motion planning problem as an algorithm to produce a continuous motion that connects a start configuration and a goal configuration [1], while avoiding collision with obstacles. The robot and obstacle

geometry is described in a 2D or 3D workspace, while the motion is represented as a path in configuration space.

1.1.1 Different Tasks or Applications of Motion Planning

Motion planning problems are not limited to the field of robotics. It has applications in various areas outside robotics as well. These include animating digital characters [5], games and camera positioning [2], architectural design, robotic surgery, molecule folding [3], and assembly/disassembly problems [4].

In [2], Dennis and Mark used a probabilistic road map motion planning method, in order to generate an automatic camera motion. The method is useful for navigation through a virtual environment where a user gives a necessary goal position and the orientation at that position. The system automatically calculates a smooth motion which connects the initial position and orientation to the goal configuration, while connecting it avoids the obstacles in the scene. A novel motion-planning algorithm can be used for character animation, presented in [5], to plan a motion through the given constrained space towards the goal configuration.

Guang and Nancy suggested an algorithm in [3] to study protein folding pathways and potential landscapes based on the motion planning method. The algorithm helps to compute folding pathways from their initial denatured state easily and effectively. An initial denatured state is the state which loses its original protein structure. Sujay, Ian and Nancy proposed an approach [4] to solve disassembly problems with help of potential field motion planning. The approach considers assembled configuration as a start configuration and the information about relative position of parts are used to generate

potential movement of the parts. In short, the information based on geometry is used to decide which parts to be moved and which not to be, at a given point of time in the disassembly process.

Motion planning is one essential part for the field of robotics. Let it be a manipulator, a mobile robot or a manipulator mounted on a mobile robot, motion planning is involved everywhere in the fields of robotics, and automation or autonomy. Various approaches to robotics motion planning are explained in the background and related work section.

In the field of mobile robotics, motion planning is essential in military application, rescue tasks [6, 7] and industrial automation/manipulation. These kinds of applications always demand a team of robots or multi-agent scenario where robots work in coordination. Tasks such as a rescue operation during a war, a bombing, earthquake or other natural or human forced disaster require multi robots in various shapes, sizes and configurations.

Multi-Link robots, sometimes called a mechanical snake, can be used for military or rescue tasks. A simple rescue scenario involving ruins of a building after an earthquake can be approached by a link-robot. Searching for victims, later on followed by a rescue operation from a site of a natural or human caused disaster is a problem that one must plan to solve in advance. Such sites are commonly inaccessible and dangerous for human workers.

In [6], authors introduce a mobile robot for rescue activities called MOIRA (Mobile Inspection Robot for a Rescue Activities). Structural features allow this robot to go in to debris or other complex environments for inspection. Hirose and Fukushima

propose a new multi-link robot called snakes and strings in [7]. This robot is able to work in rescue operations. Snake refers to a robot that can move easily and efficiently through debris. The string is useful as a power supply and a communication link. Various implementations of a mechanical snake, introduced in [7], include ACM (active cord mechanism), and the HELIX, etc.

For all these tasks and applications, a motion planning algorithm is essential. One such method is the Artificial Potential Field method.

1.1.2 Benefits of Multi-link Robot Teams

Robotic technology is essential in applications which involve hazardous worksites where human admittance is dangerous. For example, for rescue work in an earthquake affected area, bomb sites, underwater sites, nuclear power plants etc., multi-link robots can be more advantageous than conventional robots in these environments because they can deal with applications which require re-configurability as in *metamorphic* robotic systems [8]. A rescue task after an earthquake or manmade disaster, where it is often hazardous for rescue personnel to approach the sites of possible survivors demands such kind of specifically designed robots. Multi-link robots can be configured in such a way that they carry a life/pulse detector in one link, and transmitters in the other links. Whenever the robot finds someone alive, a link having a transmitter can be detached and left with the person, while the remaining links continue searching.

Increasing interest is shown in distributed robotic systems where a task is performed by a team rather than a single robot, such as *Millibot trains* [9] for industrial

applications. Proper motion planning is essential to provide for smooth and efficient motion of the robots in a tight or cluttered environment.

Despite increase overall complexity, multiple mobile robot systems have several benefits, which include:

1. Overall performance benefit compared to a single robot system
2. Several tasks or goals that are too complex or impossible for a single robot system can be achieved
3. Redundancy, reliability, fault tolerance and robustness in system
4. Can accomplish purpose built tasks

1.1.3 Artificial Potential Field Method to Solve Motion Planning Problem

Extensive amount of work has been done in the motion planning field with different authors suggesting various approaches. In this thesis, the Artificial Potential Field Method is used to approach the motion planning problem. Khatib [10], pioneered the approach for mobile robot motion planning that uses a repulsive potential field around obstacles considering a minimal safety range that forces the robot away from obstacles. An attractive potential field around the objective attracts the robot. Negative gradient vector of potential function is the total effective force that forces the robot to move downhill towards its target. This thesis presents a motion planning algorithm using an Artificial Potential Field (APF) method for an environment with a team of multi link robots, in which the robots also recover from local-minima constraint.

1.2 Problem Statement

The potential field method is popular in the motion planning area, but hand in hand, there are its limitations or drawbacks as well. The main drawback of APF is that the robot can be trapped in to local minima before reaching goal configuration. Different techniques to recover from local minima are discussed in chapter 2.

Artificial Potential field method is used for manipulators and mobile robots in various applications. Motion planning for a distributed mobile robotic system is still a complex problem. This thesis suggests solutions to the problems stated below:

- 1: To develop a motion planning algorithm for a multi-link robot teams with artificial potential field method*
- 2: To overcome the local minima problem in APF*
- 3: To model obstacles and repulsor use a function that help to improve performance of motion planning system*
- 4: Overcome the zigzagging phenomenon of gradient descent method*
- 5: Modify simulated annealing method to improvise its performance while using it along with APF for motion planning method*

1.3 Thesis Contributions

1.3.1 Multi-Link Motion Planning

An algorithm that can guide multi-link robot teams to the goal configuration is developed in this work. The algorithm is able to plan motion of the multi link robot with

help of the artificial potential field method by using Gaussian functions to model attractors and repulsors. It works in a way that the head of the link is followed by the tail of the link, considering it as a target while the head moves towards the goal configuration. The suggested method is to overcome the zigzagging phenomenon of the gradient descent method, as well as recover from local minima with help of simulated annealing method.

1.3.2 Modified Simulated Annealing (SA) for Local Minima Recovery

As the SA method chooses random neighbors, when applied to optimization problems, it may select one point or location multiple times. It is possible to reduce convergence time if we can control this random selection of neighbor in a way that it gives us best possible result at every step. Suggested modified SA selects the best candidate as a neighbor from the available random set of neighbors nearby and further rejects all the formerly visited locations while selecting the best one. In addition to that, the algorithm uses the APF method in combination with SA in a manner that once the neighbor is selected in SA domain, to reach the accepted, the neighbor robot follows the APF method and once it is at the neighbor it will come back in to the SA domain.

1.3.3 Publications from this work

A conference paper based on this algorithm was presented into an IEEE/ASME conference (MESA'10) in Qingdao, China.

- Deval Yagnik, Jing Ren and Ramiro Liscano; "Motion Planning of a Multi-Link Robot With Artificial Potential Field Method and Modified Simulated Annealing," *Proceedings of 6th MESA Conference on Electrical and Computer Engineering, IEEE ASME' 10*, Qingdao, China, 15-17 July, 2010.

1.4 Outline of the Chapters

This thesis is organized in to seven chapters. The first chapter comprises of a brief introduction to the work undertaken, introduction to the problem, and our approach.

The second chapter includes background of the problem with analysis of related work and the comparison between different approaches.

The third chapter describes the basics of the Artificial Potential Field method. Also, an analysis of the Gaussian functions used to describe attractors and repulsor, navigation function and the control law is present. A brief introduction to local minima problem is given at the end of the chapter.

In the fourth chapter, a methodology to approach multi link robot motion planning with the help of the head follower tails approach is presented. This chapter explains the reason behind the use of Newton's method instead of the gradient descent method.

The fifth chapter includes the approach to the local minima problem with an effective algorithm called simulated annealing. It describes the use of Simulated Annealing in optimization and the integration of SA and APF.

Chapter six shows the implementation of the algorithm, various simulation scenarios and results for the motion planning.

Lastly, chapter seven states conclusions and possible future work.

CHAPTER 2

Background and Related Work

2.1 Different techniques to solve motion planning problem

Motion planning is the process of dividing a task into atomic motions in order to enable a mobile robot to navigate anywhere from inside a building to a distant waypoint. Motion planning should enable the robot to execute the task effectively, by transmitting speed and turning commands to the robot. The traditional approaches to motion planning problems are summarized in the work of Latombe [11].

2.1.1 Implicit and explicit motion planning

Motion planning algorithms can be classified as explicit or implicit motion planning. Explicit motion planning explicitly computes the trajectory and actuator inputs before the motion. It divides the planning into three steps: path planning, trajectory planning and robot control. Examples for explicit methods are cell decomposition, level-set methods and road map methods. Algorithms that find the shortest path in the known graph can be used for trajectory planning as well. Widely used algorithm A-star plots an effectively traversable least cost path between points. Benefit of using this kind of

algorithm is that it is a complete method which certainly converges to solution if there is any solution available. D-star is based on A-star with dynamic constraint, and can be used for path planning in dynamic environment [12].

On the other hand implicit motion planning instead implicitly specifies interaction of the robot with the environment and the response to the information by the robot. Implicit motion planning methods such as APF are more easily implemented in an online implementation and provide a natural way to model the behavior of a multi robot system. APF combines all three steps of motion planning (path planning, trajectory planning and control law) in to one. In addition to that APF is a local path planning method that considers only local information while planning the motion of the robot which in turn reduces the computational time.

2.1.2 Intelligent techniques and Control theory based techniques

With the introduction of intelligent techniques, such as fuzzy logic based, neural network based, genetic algorithm and behavior-based methods, the motion planning field is divided into two main groups. These are intelligent techniques [13 - 21], and control theory based techniques, such as Artificial Potential Field, Road Map and Cell Decomposition Methods [10, 22- 25]. Some authors have tried to approach the problem by combining both the control theory based and intelligent based techniques [26 - 27].

The neural network based approach is a computational model used for optimization problems and it is based on the biological neural network structure. The most attractive feature of this approach is its learning ability which makes it a

competitive candidate for real life applications. Autonomous robot operations need integration of sensing and motion planning. Perceptual Control Manifold (PCM) is a concept which considers sensors as a part of motion planning. In [18] Zeller et al. proposed an algorithm base that uses a self organizing neural network to learn the topology of manifold rather than the analytical approach. Jing Yuan designed a collision identification neural network for collision free motion planning in 3-D space [19].

Robot path planning problems can be a part of the known or unknown environment. One can't always obtain a precise model of the system. Hence, fuzzy logic controllers are a convenient choice to find optimal paths as they incorporate heuristic knowledge in form of if-then rules. S. Boonphoapichart represents a fuzzy logic based approach to motion planning for an uncertain environment with multi-objective goals in [20]. In [13], authors introduce a fuzzy tournament selection method that can be used with the evolutionary path planning method.

Genetic optimization is time consuming so it is attractive mainly in offline planning rather than online path planning. A global path planner for a large space mobile robot application based on genetic algorithm was proposed by Gerke in [15] which works offline with the known environment. Path planning using the collision avoiding approach is based on searching the collision free path throughout the work space. Baba and Kubota introduced a collision avoidance approach, where to generate collision free paths they used genetic algorithms [16].

Ren and Tse [21], proposed an approach with behavior programming to avoid disturbances from internet latency for an event driven mobile robot system. The event driven approach is applied on the mobile robot to switch the behaviors in order to execute

the uncertain assignment autonomously. These artificial intelligence methods are related to the use of optimization algorithms for optimal global path planning that are relatively complex and take excessive computing time.

The roadmap method works on capturing the connectivity of the robot's free space in network of one-dimensional curves. These curves are called roadmap, and they are lying in the free space of the robot's work space like webs. The constructed roadmap is used as a set of standardized paths, so when a motion planning problem is presented it is reduced to connecting the start and the final goal configuration to the points already there in roadmap, and in finding the connecting path from roadmap between these points. So the final path will be starting from initial configuration to roadmap, part of the roadmap and the path connecting the roadmap to the goal position. Various methods based on roadmap have been proposed so far, such as visibility graph, voronoi diagram, freeway net and silhouette. A probabilistic roadmap planner is discussed in [25] where a probabilistic roadmap is constructed using local planner and then according to start and final configuration query is to be made.

Cell decomposition methods are also popular to approach motion planning. The idea behind this approach is to decompose the free space of robot work space into simple and smaller regions called cells. These cells subdivide the free space such that the path between two points in a cell can be easily generated. Once the free space is decomposed in small cells, a connectivity graph is constructed with respect to the adjacency relationships between the cells in which the nodes represent the cells, and the links between the cells show that they are adjacent to each other. In this way, by following

adjacent cells from initial configuration to the final configuration we can figure out a continuous channel.

The first type of cell decomposition method is exact cell decomposition where the free space is decomposed simply by parallel line segments from each vertex of interior obstacle boundary to the exterior work space boundary. Each cell is numbered as a node, and adjacent nodes are linked in a sequence of cells that make channels in free space. It contains the free path by connecting the initial and goal points. On the other hand, approximate cell decomposition uses a recursive method to continue the subdivision of the cells until each cell lies completely in free space or completely in obstacle region. Once the limit of resolution is reached further decomposition would not be in effect. Each time a cell is divided in to four same parts. After the decomposition step, by following the adjacent cells through free space we can find the free path. In [24], the concept of cell decomposition is combined with probabilistic sampling where an approximate representation of the collision free path is obtained in to the free space and used to guide probabilistic sampling over that representation.

One of the essential control theory based technique is Artificial Potential Field (APF). Since its conception in 1983, Potential Field Approaches are widely used and expanded for motion planning due to its inherent simplicity [28] and elegance. APF consumes less time and computational power being a local planning method it plans the motion in a way that at every step it cares about the obstacles in the vicinity of a robot.

Many authors have used two or more methods to approach the problem such as [26-27]; mainly an intelligent method in combination with APF. In [26] Plumer proposed an approach where a feed forward neural network is used to control the steering of a

vehicle using local information generated from APF. Kun et al [27], proposed an adaptive fuzzy controller to plan the path that improves the flexibility of the APF method. The proposed algorithm will switch to a fuzzy tracking mode when the robot is trapped in local minimum and uses genetic algorithm to adjust the fuzzy control rules.

2.1.3 Artificial Potential Field Method

The potential function that was proposed by Khatib [10] is given by,

$$U_{APF}(x, y) = U_{x_d}(x, y) + U_o(x, y) \quad (\text{Equation 2.1})$$

Where, $U_{x_d}(x, y)$ is the attractive potential and $U_o(x, y)$ is the repulsive potential and they can be defined as,

$$U_{x_d}(x, y) = 0.5 k (x - x_d)^2 \quad (\text{Equation 2.2})$$

And

$$U_o(x, y) = \begin{cases} 0.5 \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right)^2, & \text{if } \rho \leq \rho_0 \\ 0, & \text{if } \rho > \rho_0 \end{cases} \quad (\text{Equation 2.3})$$

Here $(x - x_d)$ is the distance between robot current at that instant to the goal configuration, k is a scaling constant which specifies the strength of attractive potential, $\rho(x)$ is distance to the obstacle, ρ_0 is minimum distance from current configuration to the point where robot touches the obstacle, and η is a scaling constant specifying the strength of repulsive field.

A significant amount of research has been conducted using APF since its introduction by Khatib. Different potential functions have been designed for the approach

such as super-quadric functions [29] and harmonic functions [30], Khatib's FIRAS function [10], Ge and Cui's new potential function [31] and hydrodynamic function [22] to model objective and obstacles. These are local approaches as only local gradient information is required to compute these functions. Being an implicit approach, they do not require any kind of process before actual movement and hence are seen advantageous from a computational point of view. Detailed analysis of all these different potential functions and survey of development in this field is summarized by Leng Feng Lee in [32] and by Ren in [33].

2.2 Multi-Link Motion Planning with APF

Even though there are many successful approaches presented to solve single robot motion planning, they can't be applied to the motion planning problem for cooperative multi-robot teams. Motion planning for multi-link robot teams consists of several problems including coordination strategy [34], mutual collision avoidance between two robots from different or the same link, and task scheduling. In [34], collision avoidance between robots in multiple mobile robot application is discussed.

Extensive research work has been done in the field of distributed robotic system or multi robot system. In applications such as military operation, rescue task and industrial manipulation where task needs to be divided between robots, multi robot system is essential. A distributed system gives us various benefits ranging from execution of specific purpose built tasks that can't be done with a single robot, to advantages like robustness and redundancy. One such example is swarm robots developed at the University of the West of England [35]. The method to be developed has swarms of

relatively simple robots team up to show emergent intelligence that is greater than the sum of all the individual robots.

2.2.1 Head Follower Tails with APF

Motion planning for multi link mobile robots is a complex task, approached in different ways. In the suggested method, the first robot of the link, considered as a head, follows gradient of the potential function while considering the goal as an attractor and avoiding obstacles. Successive robots are considered as a tail, and follow the robot ahead of them in the link considering it as an attractor. As shown in Figure 3, robot 3 follows robot 2 considering it as an attractor. Two forces, an attractive force in the direction of robot 2, and a repulsive force which forces away from obstacle, act on the robot. The resultant direction is the synthesis of all the effects as shown in the figure which allow the tail to move away from the obstacle, as well as travel towards the goal.

We model the attractors and repulsor with Gaussian function to form the attractive and repulsive fields. This is advantageous compared to other functions described in chapter three. In addition, for all link robots in the system we used Newton's method to replace gradient descent, or the steepest descent method, which is used for a typical APF algorithm. Notably, Newton's method allows faster convergence, more speed and eliminates zigzagging that occurs in the steepest descent method.

This approach can be used in a way that the robots can work in coordination with each other or individually with slight changes in algorithm. An example of such a system is described in [36] where each robot carries a manipulator that can be used for either individual or a defined cooperative task. They can connect with each other physically or can be separated according to the requirements of the task. The algorithm can be used in

a large distributed robotics system where a number of robots can work together for a specific task while moving in formation.

The rationale behind using APF is that it is a simple, efficient and fast method of path planning. The elegance and efficiency of a real snake's movements can hardly be reproduced by a robot but it's still a task to learn from nature. An observation of a snake's path in getting around an obstacle will reveal that the whole body follows the trace of the head. Head avoids obstacles in influence of the repulsive potential field. In the method developed, the head moves towards the target while subsequent links follow the head avoiding any obstacles.

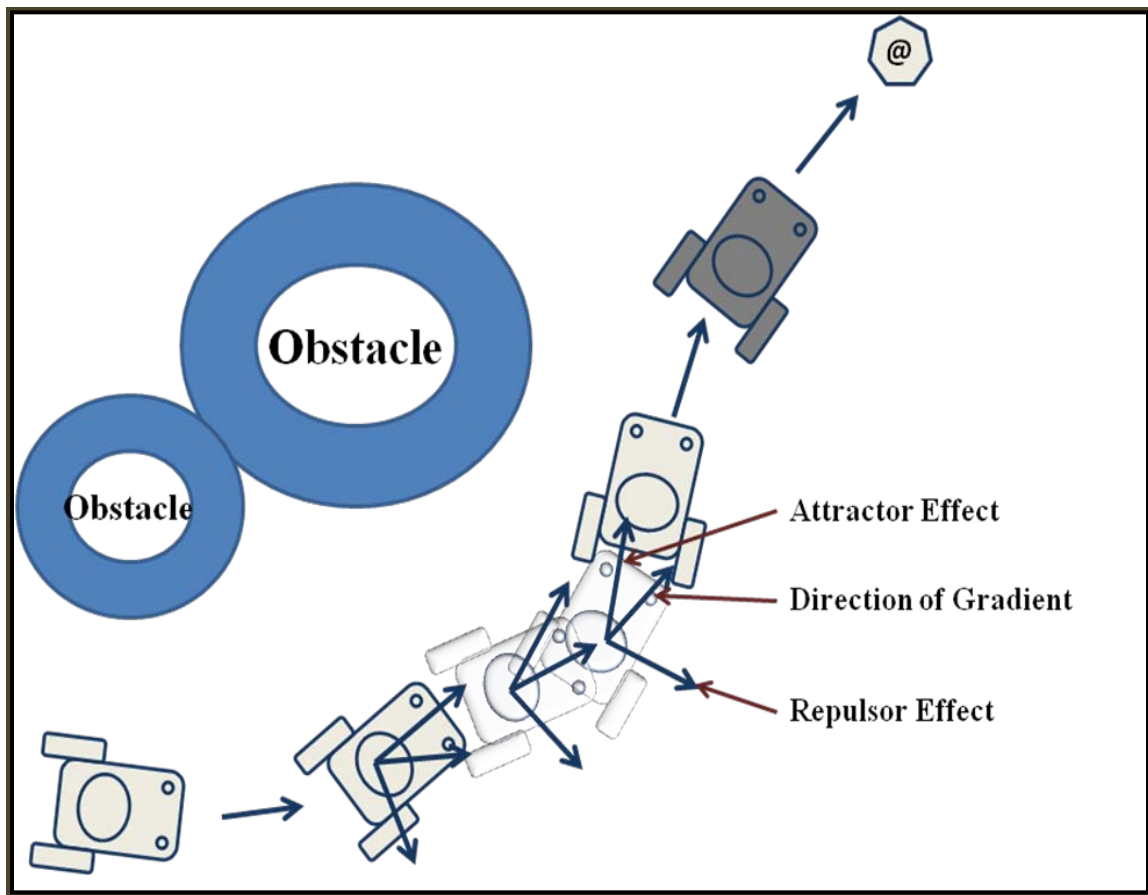


Figure 3 Multi-Link Robots Navigation

2.2.2 Gradient Descent Method and Newton's Method

Ren, McIssac and Huang suggested in [37] that use of Newton's method to replace the gradient descent approach can be done in APF to improve system performance while moving in the vicinity of obstacles. Typically, control laws for potential field systems generate motion along gradient directions which means it follows steepest descent. The steepest descent approach has a few drawbacks including rapid change in direction (zigzagging) when in the vicinity of an obstacle. They generate so many controls that are not necessary which reduces the convergence time by increasing the number of moves. Newton's method allows faster convergence, more speed and eliminates zigzagging.

2.2.3 Limitations of APF Method

The inherent problems in this APF method are described by Koren and Borenstein in 1991 [38] with help of mathematical analysis. The four main problems identified in the method are:

1. *Trap situation for the mobile robot because of Local Minima*
2. *Narrow corridor between closely spaced obstacles*
3. *Oscillations in presence of large obstacle*
4. *Oscillation in movement while going through narrow passages*

Replacement of the gradient descent method with Newton's method helps to reduce the oscillation in robot movement [37]. Use of Gaussian function to represent the obstacles and attractors helps us to adjust the affecting areas of obstacles which results in

the extra advantage to reduce the problem caused by the narrow passage between closely spaced obstacles. So far the major drawback of the potential field method for us remains the existence of local minima which can trap the robot in an undesirable position before reaching the goal.

2.3 Local Minima Recovery

So far researchers tried to approach the problems such as local minima inherent with the Potential Field Method in different ways. In [39], P Vadakkepat et al., suggest escape-force algorithms to recover from local minima. In the paper, the author describes an Evolutionary Artificial Potential Field where APF is combined with genetic algorithms to derive optimal potential field. To recover from local minima this method uses additional algorithm which is the escape force algorithm.

In recent time, S. S. Ge and Y. J Cui proposed an approach to alleviate a GNRON problem, and reduces local minima problem by a redefinition of the potential functions which gives fewer local minima occurrences [31]. Another way is to use an efficient search technique, such as Simulated Annealing, which has the capability of escaping from local minima in APF with the help of uphill moves. As local minima problems exist in multi-link robot scenarios, an approach of SA with APF is used to enable the robot to recover. Simulated Annealing is to date the widely accepted algorithm for finding a global minimum of an optimization function.

2.4 Proposed Approach to Local Minima Problem

In 1982, three researchers from IBM [Kirkpatrick et al., 1983], suggested a new iterative method of simulated annealing that can avoid local minima in a difficult optimization problem [40]. Simulated Annealing is used for various applications like image processing, circuit design, path planning problem etc. It's a robust algorithm that allows uphill moves which in turn avoids local minima. Simulated Annealing (SA) is a Generic Probabilistic Metaheuristics which is described by Bertsimas and Tsitsiklis [41] as a probabilistic method for finding the global minimum of a cost function which may pose several local minima. It is arguably one of the most robust and effective methods to recover from the local minima constraints in optimization problems [42]. A combination of Simulated Annealing with the Artificial Potential Field method gives the optimal solution to the complex motion planning problem. Later chapters will explain how SA and APF are combined to solve motion planning problems. An extensive effort has been made to utilize the SA technique for path planning [43]. Local minima problem occurs in multi link robot systems working under the APF method as well, which leads us to use SA.

As the SA method chooses random neighbors, as applied to optimization problems, it may select one point or location multiple times. Our modified SA selects the best candidate for a neighbor from the available random set of neighbors nearby and further rejects all the formerly visited locations while selecting the best one. Each time, the algorithm sets a new local goal and continues until the robot escapes from local minima. In addition, we are using the APF method in combination with SA such that

while in SA domain for the movement to the accepted the neighbor algorithm will use the APF method.

Simulated annealing can be probabilistically complete if its probability of finding a solution can be made to 1. In that case, a random search algorithm will need a long computation time. Furthermore, global algorithms use all the information about the environment to plan a motion from the start to goal point. Local algorithms, like APF, are made to avoid the obstacles that are in the vicinity of the robot and they need information related to those nearby obstacles only [44]. This makes them computationally very efficient and less time consuming which in turn proves their ability to work well in real life systems. This is one of the reasons why we prefer APF for motion planning of a known environment.

2.5 Assumptions of Robot Capabilities

For development of the algorithm, we assume several robot capabilities that can be easily implemented on mid-sized mobile robots like Koala. We assume the robots to have knowledge of their position, knowledge of the obstacle's locations, and the ability to communicate with the centralized system. For positional information robots can use devices, like the GPS while the workspace can be mapped with help of camera sensors. For communications, robot can use the Bluetooth module to send/receive location information to/from the knowledge base and receive control signals related to the movement. We need to incorporate sensing into the system to relax some of the assumptions. On the other hand while adding sensing one need to consider the error, noise of sensing and communication system as well.

CHAPTER 3

Artificial Potential Field and Navigation Function

3.1 Artificial Potential Field

Artificial potential field methods can be implemented quickly in the real world, and are more suitable to real life applications as they require only local gradient information. The potential field methods are popular in the context of obstacle avoidance applications for both mobile robots and manipulators since Khatib [10]. The principle of the approach is specified by using imaginary forces acting on a robot. Obstacles in the work space have repulsive fields around them and the objective has an attractive field. In the context of motion planning, minimum potential is at a goal configuration that makes the objective a global minimum of the overall work space. The synthesis of all the attractive and repulsive forces determines the direction and speed of the robot's movement [45].

3.1.1 Gaussian Models

As defined by Ren and McIsaac in [46], the proposed method uses a Gaussian function to model attractors and a higher order Gaussian like function to model obstacles. Use of the Gaussian functions reduces the number of local minima occurrences as well.

3.1.1.1 Gaussian Function for Attractors

We have used potential fields with two dimensional Gaussian functions to model attractors and obstacles. Targets or goal location located at (a_x, a_y) coordinates are represented by the negative Gaussian attraction function [46]:

$$F_A(x, y) = 1 - e^{-\left(\frac{(x-a_x)^2 + (y-a_y)^2}{2\sigma^2}\right)} \quad (\text{Equation 3.1})$$

Here the value of the variance σ is the measure of the area of effect for the attractor. This value is generally higher for the attractor so it covers the whole work space. Figure 4 illustrates this fact clearly with contour plots for goal configuration over the work space.

Gradient of $F_A(x, y)$ will be used to calculate the gradient of navigation function,

$$\nabla F_A(x, y) = \begin{bmatrix} \nabla_x F_A \\ \nabla_y F_A \end{bmatrix} \quad (\text{Equation 3.2})$$

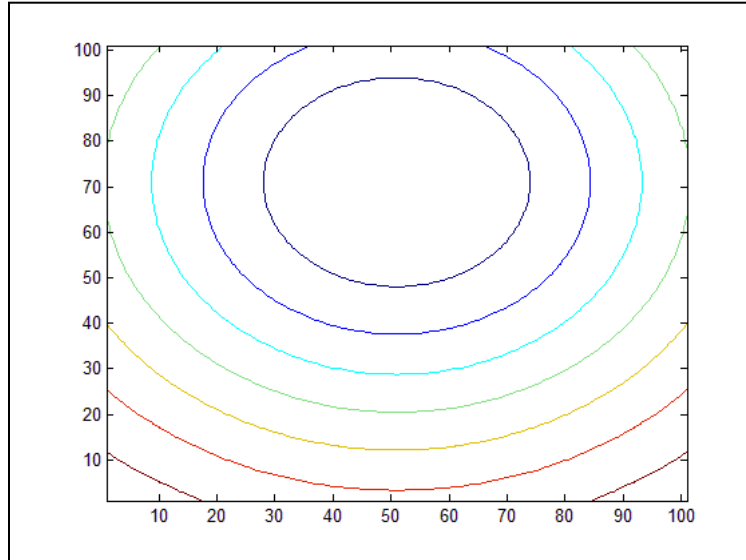


Figure 4(a)

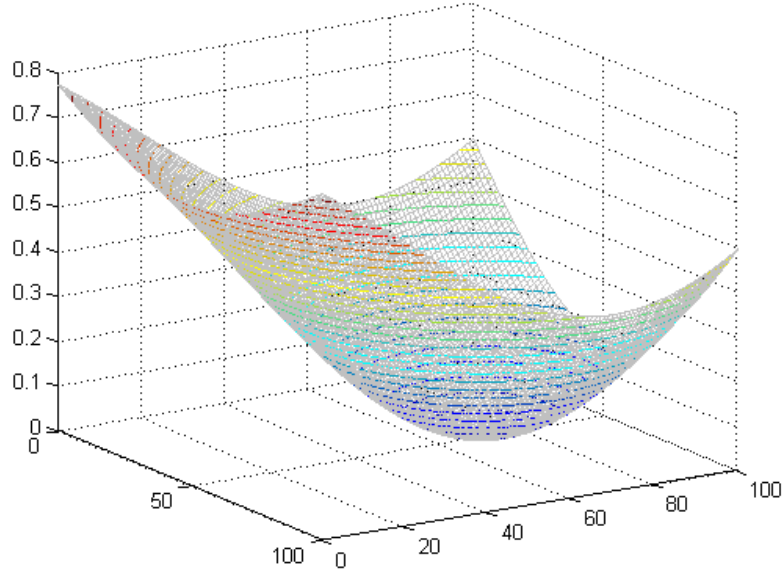


Figure 4(b)

Figure 4 (a-b) Contour plots of negative Gaussian function for goal configuration over the work space in 2-D and 3-D

3.1.1.2 Gaussian Function for Obstacles

Obstacles and robots located at (r_x, r_y) are treated as repulsive fields and modeled with the circular two dimensional Gaussian like repulsion function [46]:

$$F_O(x, y) = e^{-\frac{1}{2} \left(\frac{(x-r_x)^2 + (y-r_y)^2}{\sigma^2} \right)^C} \quad (\text{Equation 3.3})$$

Here, the variance σ is the measure of the size of the obstacle and the integer C determines the effective range of the obstacle. Convex obstacle shapes are represented by super-scribed circles in our method.

Gradient of the function is:

$$\nabla F_o(x, y) = \begin{bmatrix} \nabla_x F_o \\ \nabla_y F_o \end{bmatrix} \quad (\text{Equation 3.4})$$

For obstacle function, the value of variance and integer C will be set according to the requirement of the task or user which in turn localizes the effect of the repulsor. Three sparsely spaced obstacles with the function in equation 2 will create contour plots shown in Figure 5.

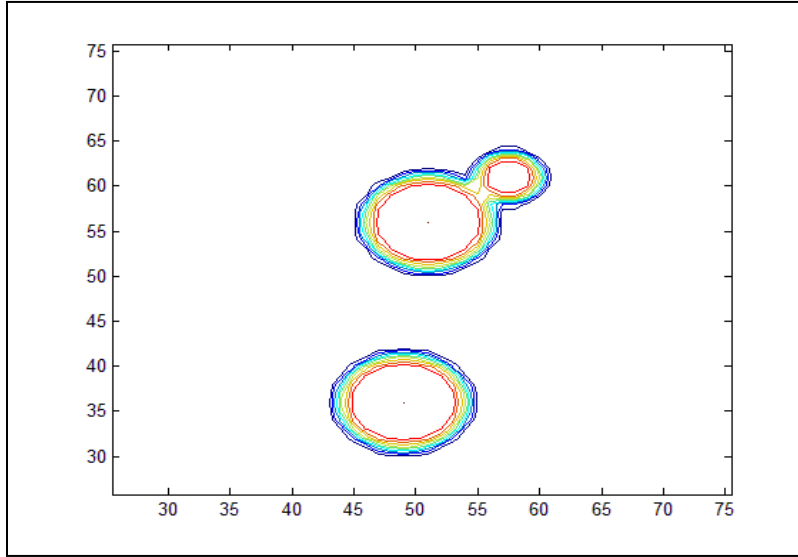


Figure 5(a)

With Gaussian functions we can model obstacles of different shape with approximate details of their size and location. With the approximate location and size we can model the super scribed circle of a certain diameter that covers the obstacle which helps us to model obstacles even if we don't have very much precise shape and size details.

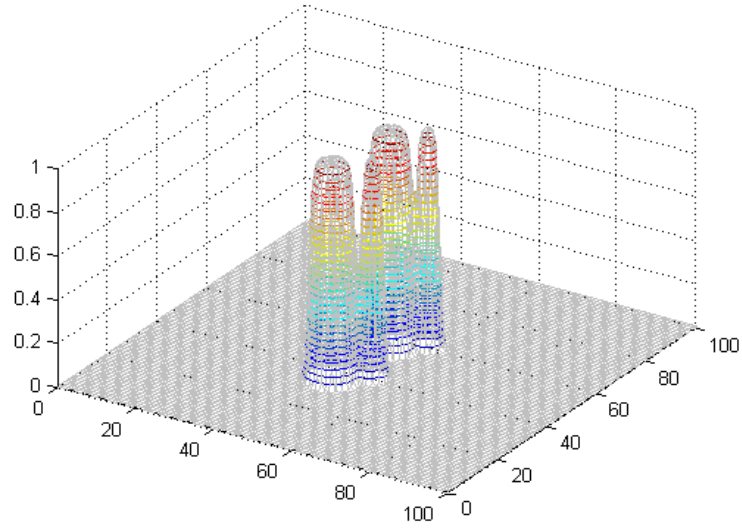


Figure 5(b)

Figure 5(a-b) Contour plots of Gaussian function for obstacles in 2-D and 3-D over the work space

3.1.1.3 Model of Workspace

To model the attractor and obstacles we used Gaussian functions because of the flexibility while modeling and to reduce the overall occurrences of Local Minima in the work space with use of super scribed circles [33],

- 1) The affecting range (steepness) of the repulsor or attractors can be controlled by variance σ according to their size.
- 2) The effect of repulsive obstacles can be localized according to requirement by changing the parameter C . That will help us to highly localize the effect of repulsion, which in turn help us to overcome the GNRON problem. The problem will be discussed later in details.

- 3) By controlling the affecting area and range we can specify minimum clearance between two different obstacles according to the size of the robot.

This will result in fewer possible local minima occurrences.

Four sparsely placed obstacles and a single target forms a workspace as shown in figure 6 and figure 7 if we model each attractor and repulsor with these Gaussian like functions. The target being a global minimum of the configuration space has an optimum value of navigation function over the workspace. For a multi-link robot each robot behaves as an obstacle for the others generating a repulsive field in addition to the fixed obstacles of the work space.

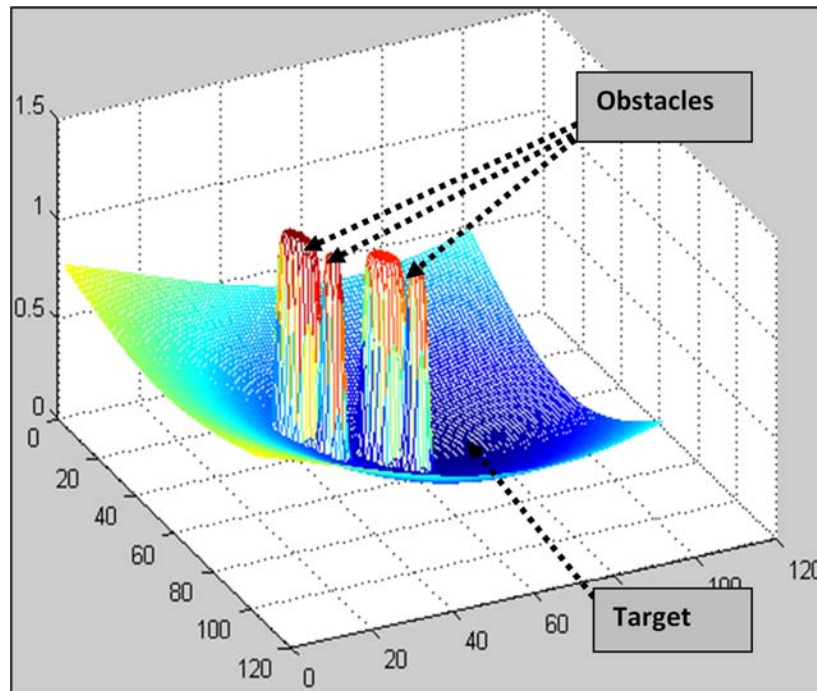


Figure 6 Workspace model with a single target and four obstacles generated with a Gaussian function

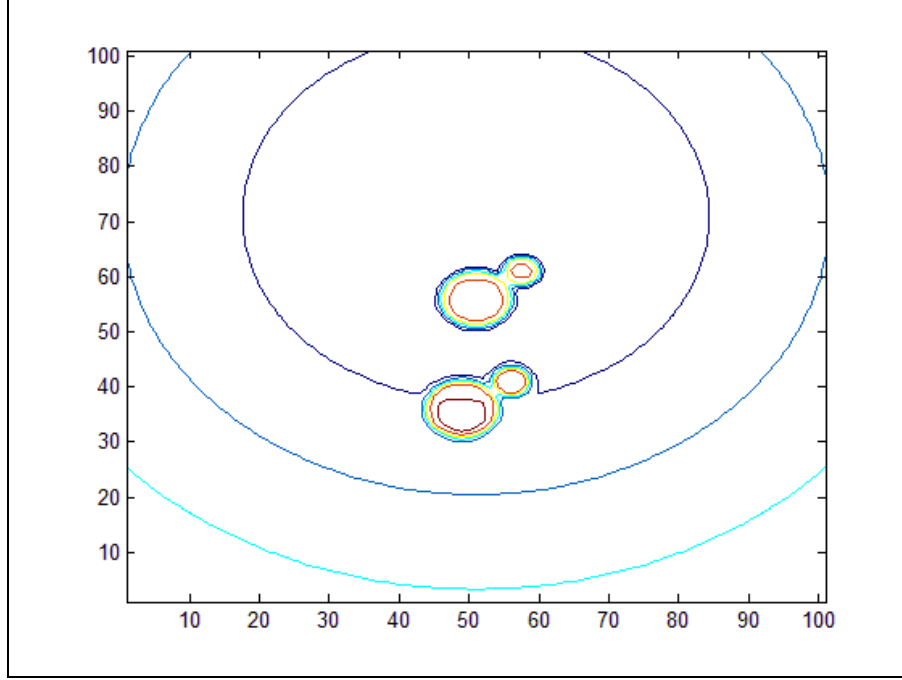


Figure 7 Contour plots for the goal and 4 obstacles in the workspace

3.1.2 Navigation Function

Motion planning needs a navigation function $F_i(q)$ for each robot i . The navigation function depends upon Gaussian-like functions for each obstacle, target and other robots as shown in equation (3.5).

$$F_i(q) = FA(q_i) + FO(q_i) + \sum_{i \neq j} FR(q_i, q_j) \quad (\text{Equation 3.5})$$

Where $FA(q_i)$ represents the sum of the effects on link robot i of all the attractors in the work space during movement. N_A is the number of attractors, taken as 1, because each robot is having a single target location in our case. For a search task or rescue task where the robots need to visit more than one goal position, N_A changes accordingly. The value of $FA(q_i)$ can be given by:

$$FA(q_i) = \sum_{k=1}^{N_A} \left(1 - e^{-\left(\frac{(x_i - (a_k)x)^2 + (y_i - (a_k)y)^2}{2\sigma^2} \right)} \right) \quad (\text{Equation 3.6})$$

$FO(q_i)$ represents the sum of the effects on robot i of all the known fixed obstacles in the work space and N_o (number of fixed obstacles) will be constant. In case of an uncertain environment where moving obstacles come into effect, the value of N_o will be changed. $FO(q_i)$ can be represented by:

$$FO(q_i) = \sum_{k=1}^{N_o} \left(e^{-\frac{1}{2} \left(\frac{(x_i - (r_k)x)^2 + (y_i - (r_k)y)^2}{\sigma^2} \right)} \right)^c \quad (\text{Equation 3.7})$$

The third part of the equation (3.5) is $FR(q_i, q_j)$, which represents the repulsive effect between pairs of robots i and j . The number of robots is assumed to be constant,

$$FR(q_i, q_j) = e^{-\frac{1}{2} \left(\frac{(x_i - x_j)^2 + (y_i - y_j)^2}{\sigma^2} \right)}^c \quad (\text{Equation 3.8})$$

3.1.3 Control Law

The control law defined below represents the dynamics of the robot. This is obtained by differentiating the navigation function described in equation (3.5).

$$\dot{q}_i = -\alpha \left(\frac{\partial F_i}{\partial q_i} \right) \quad (\text{Equation 3.9})$$

In equation (3.9), $\frac{\partial F_i}{\partial q_i}$ represents the gradient of $F_i(q)$ with respect to q_i , and α determines the speed of movement of the robot and behaves like a gain factor. The value of α can be set according to the requirements of the robot's movement. The speed of

movement should not be reliant upon position. Since the function used here decays rapidly, a unit gradient is used.

As shown in figure 8, \dot{q}_A and \dot{q}_R are the applied velocities resulting from attraction and repulsion potential functions respectively. Direction of resulting velocity \dot{q} for the robot surrounded by the obstacle and goal as shown in Figure 8 is the vector sum of \dot{q}_A and \dot{q}_R ,

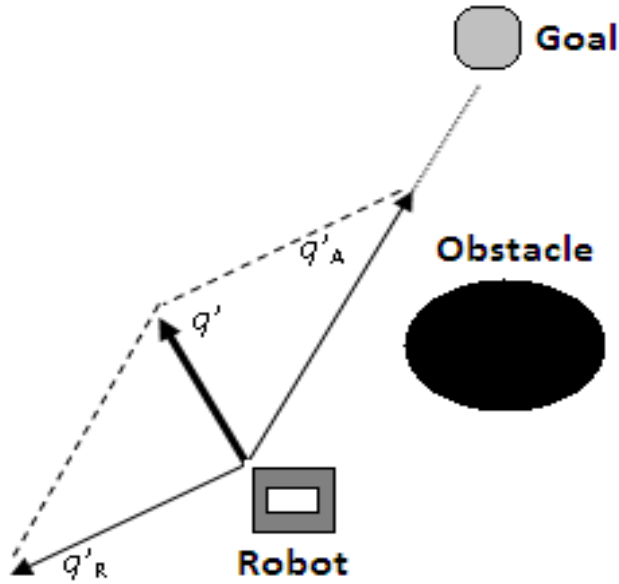


Figure 8 Vector diagrams of velocity enforced by attracter, by obstacle and the resultant velocity

3.2 Limitations of APF

The inherent limitations in APF methods are mainly local minima, GNRON (goal non-reachable with obstacle nearby), no passage between closely spaced obstacles and oscillations in presence of obstacles and in narrow passages.

3.2.1 Local Minima

With the help of super scribed circles over the obstacles in the proposed APF, which uses Gaussian functions to model obstacles and goal configuration one can avoid some of the local minima occurrences. As shown in Figure 9, a single obstacle can create a local minima situation because point C has a higher attractor potential compared to A and B. With adjustment of the integer value and variance, one can create a super scribed circle of Gaussian functions over the obstacle shape. As shown in the right side of the figure, with Gaussian functions and super scribed circles, the value of attractor potential at C is now higher than A and B which in turn avoids local minima occurrences.

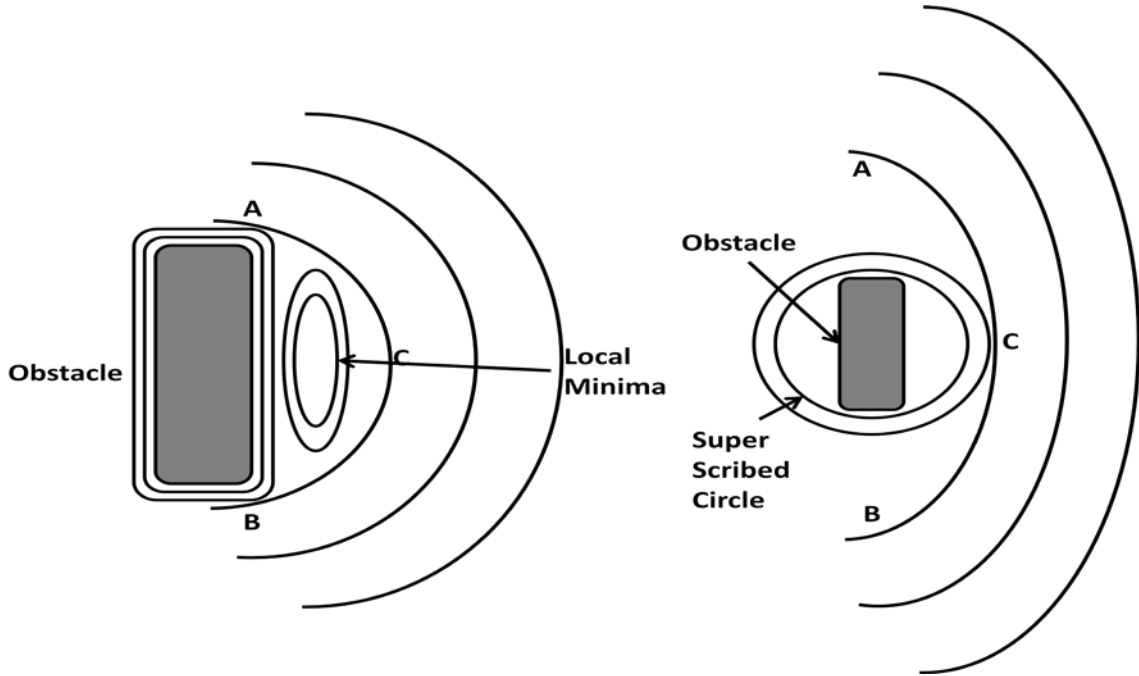


Figure 9 Gaussian Function to avoid some of the undesired local minima

In a complex environment, having more than one obstacle, Local Minima is a tough and challenging problem in many ways. One such scenario is shown in Figure 10(a-b), where two obstacles are creating local minima. Between two closely spaced

obstacles a local minima resides which can be seen as a closed loop of the attractor contour curves.

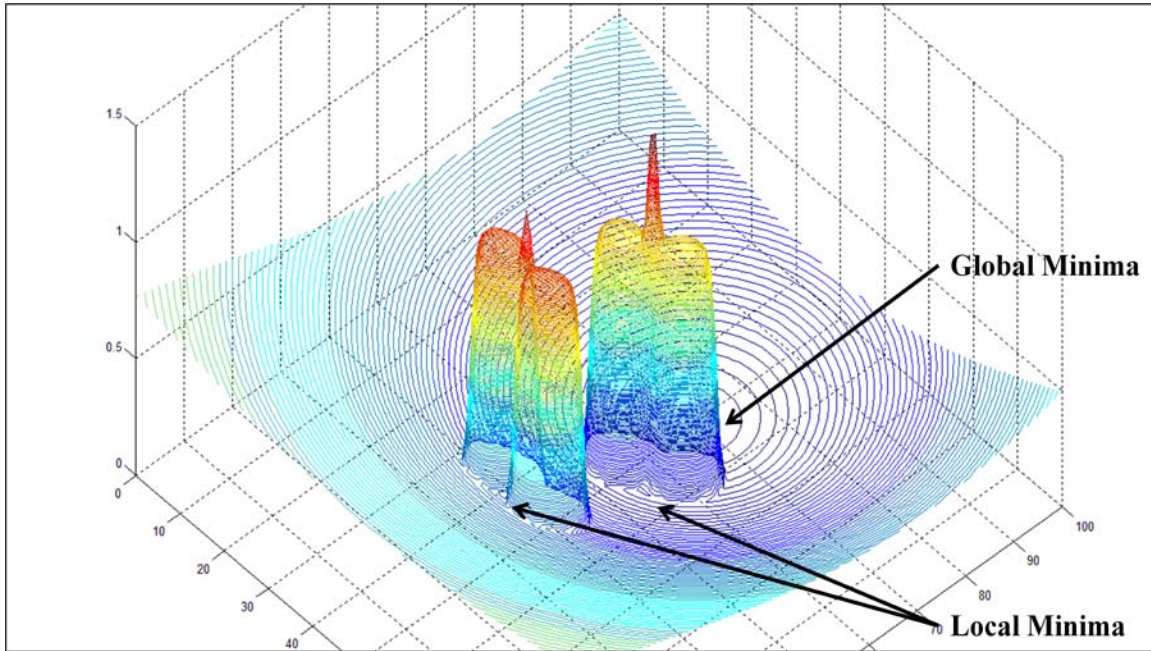


Figure 10(a)

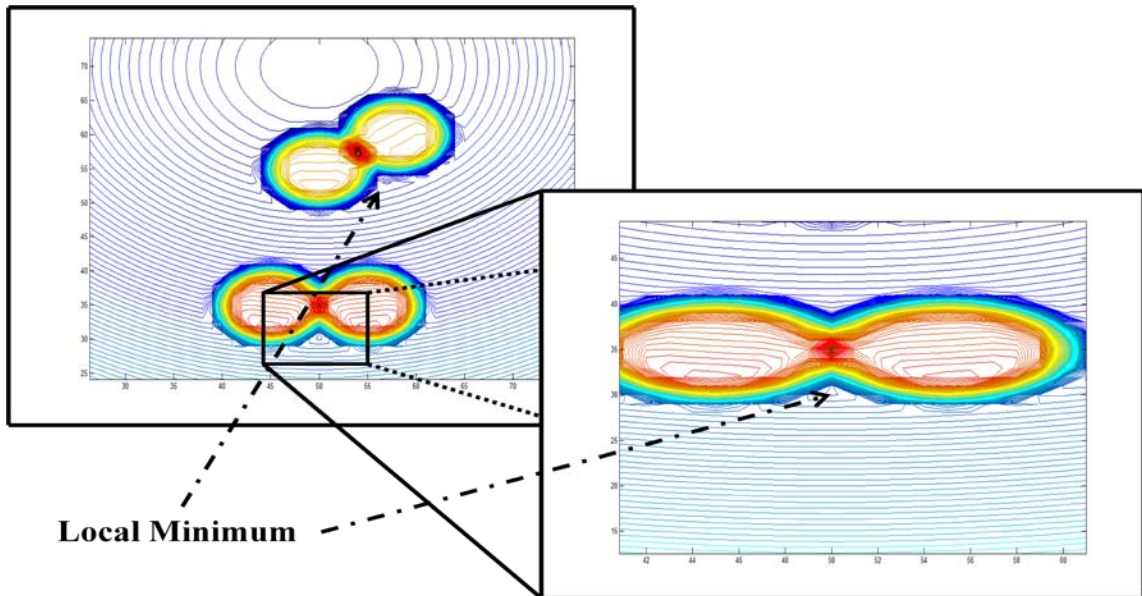


Figure 10(b)

Figure 10(a-b) Local Minimum in multi obstacle work space

3.2.2 GNRON Problem

In this scenario, where the goal configuration is very close to the obstacles, the potential field at the goal position can be changed by a high repulsive force nearby. As a result of this effect, goal configuration will no longer be a global minimum of the overall potential as illustrated by the Figure 11(a-b). Goal configuration is shown as @ in this figure which is not at the center of the contour plot for the attractors because of the positive Gaussian functions of obstacles nearby.

With the help of the generalized Gaussian functions one can change the value of variable C in the Gaussian function for obstacles which in turn will localize the effect of repulsive obstacles. As illustrated by figure 12 the effect of the positive Gaussian functions can be localized and range of obstacles can be adjusted in a way that one can overcome the GNRON problem.

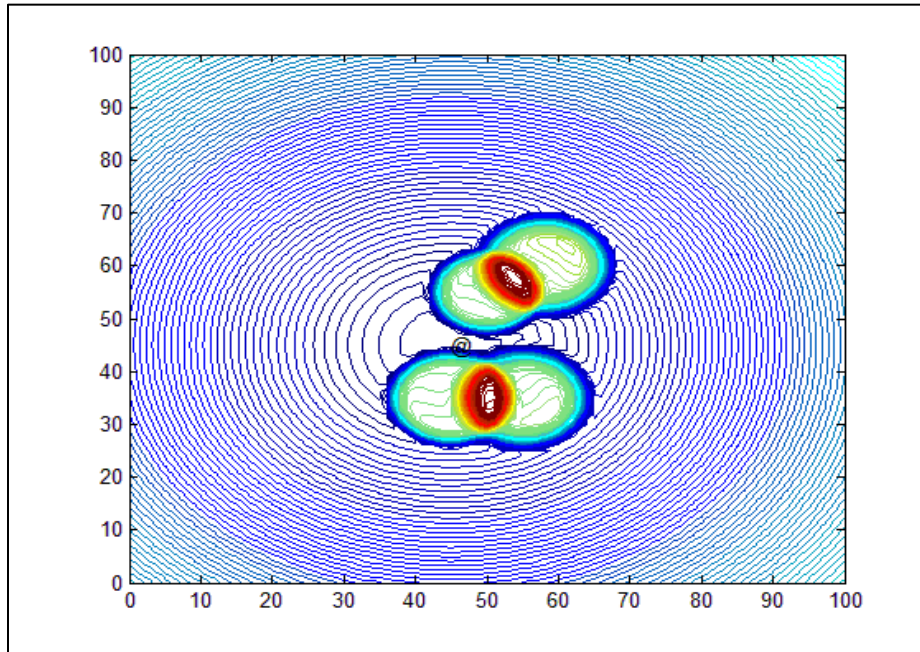


Figure 11(b)

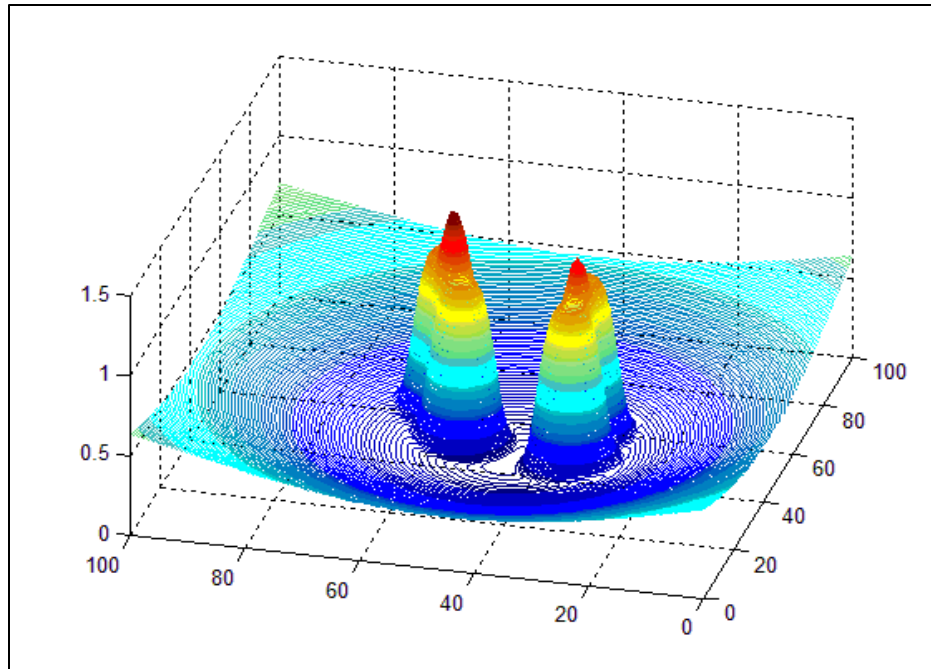


Figure 11(a)

Figure 11 (a-b) GNRON Problems in APF

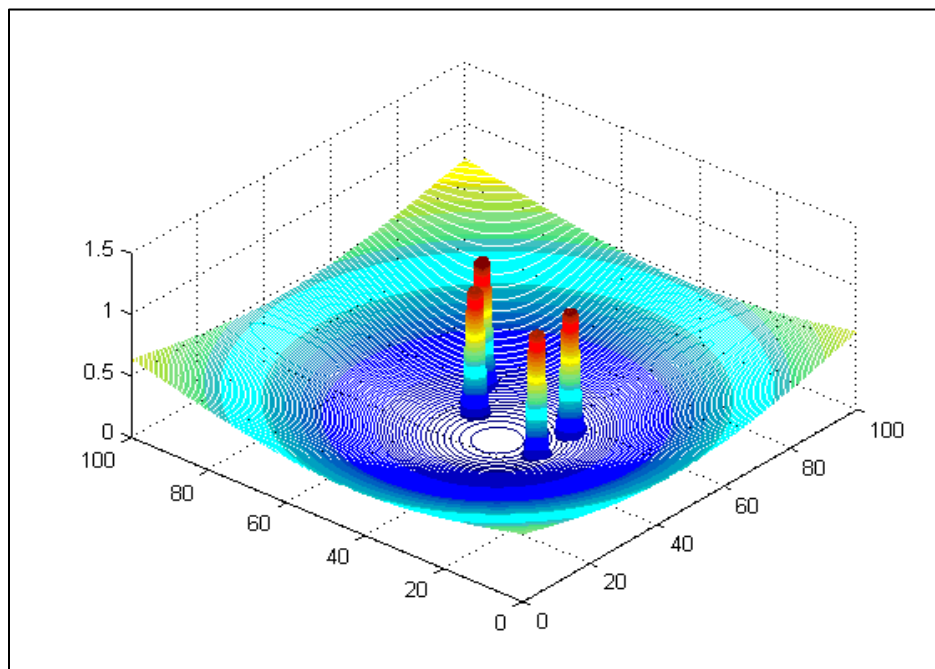


Figure 12 GNRON Problem solutions with Gaussian function

Figure 11 and 12 are having obstacles at same places, in later figure the obstacle's effect is highly localized which avoids GNRON problem by allowing the goal configuration to be a local minimum of work space.

3.2.3 No Passage between Closely Spaced Obstacles

In case there are two closely spaced obstacles in the work space, it is possible that the robot cannot pass through the passage between them. This is because of repulsive forces might be larger than the attractive force in the passage. To a certain amount this problem can also be overcome by localizing the effect of obstacles. One can increase the value of the attractive force by changing the gain factor in this kind of a situation if the size of the real robot is smaller than the passage in real world application.

3.2.4 Oscillations in the Presence of Obstacles and Narrow Passages

Repulsive force of obstacles will be prominent when the robot is moving in the vicinity of the obstacles. This will develop oscillations in the movement mainly because of the gradient descent method. Similarly, due to the repulsive force from obstacles, both the sides, while moving through narrow passages, develop oscillations in robot movement. By replacing the gradient descent method with Newton's method, these oscillations can be controlled. To further reduce the effect of the oscillations, one can decrease the value of the gain factor to reduce the speed of the robot in this kind of a situation.

CHAPTER 4

Multi-Link Robot Navigation with APF

The method proposed by Ren and McIssac, with the Gaussian function modeling in APF approach, is extended in a manner that a team of robots can work in to formation, such as multi-link, where successive tails will follow the head considering it as a target. In addition, the next chapter will show how the algorithm helps robots to recover from local minima.

4.1 Head Navigation

The robot head is guided to its goal position by following the negative gradient of the potential field generated with help of the steepest descent method. As shown in the Figure 8 and equation (3.7) $\left(-\frac{\partial F_i}{\partial q_i}\right)$ of control law, the head follows the negative gradient of the navigation function, avoiding obstacles while reaching the target location. We observe the negative gradient to be velocity vectors. Since the robot trails the path of negative gradient downhill, this method is a gradient descent or steepest descent method.

4.1.1 Gradient Descent Method

Gradient Descent is a first order optimization algorithm typically used to find global minimum of any function. In APF methods, gradient descent is used to find the global minimum of the navigation function to reach the goal configuration. To reach the

global minimum, that is the goal location, the robot needs to take steps proportional to the negative of the gradient of the navigation function at every point.

Equation 3 shows the navigation function $F_i(q)$, which decreases the fastest if we move from q_i in the direction of negative gradient shown in equation 3.7 at q_i .

$$q_{i+1} = q_i - \alpha * \left\{ \frac{\partial F_i}{\partial q_i} \right\} \quad (\text{Equation 4.1})$$

Here α represents the step size that in turn decides the speed of the robot movement. The value of step size will allow change in the value of navigation function at each step so that the navigation functions at each step will decrease [36].

The gradient descent method is relatively slow closer to the minimum, and it also has the zigzagging phenomenon that further increases convergence time.

4.1.2 Newton's Method

To overcome the disadvantages of the gradient descent method, we replaced it with Newton's method [37]. Here, rather than following the negative of the gradient, we are using a second order derivative. Newton's method is used to navigate not only the head but all the successive tails as well. By differentiating the navigation function $F_i(q)$ twice we can define a hessian matrix of the function. Iterative scheme to reach the global minimum can be generalized by replacing the gradient with the inverse of the hessian matrix. With inclusion of the positive definite matrix derived from hessian in to the control law we can overcome the zigzagging, and force the robot to move smoothly around the obstacle. As shown in Figure 13, we can get much better results with

Newton's method. Green lines show the trace of the head with the gradient descent method, while the red line shows the trace of the head when following Newton's method.

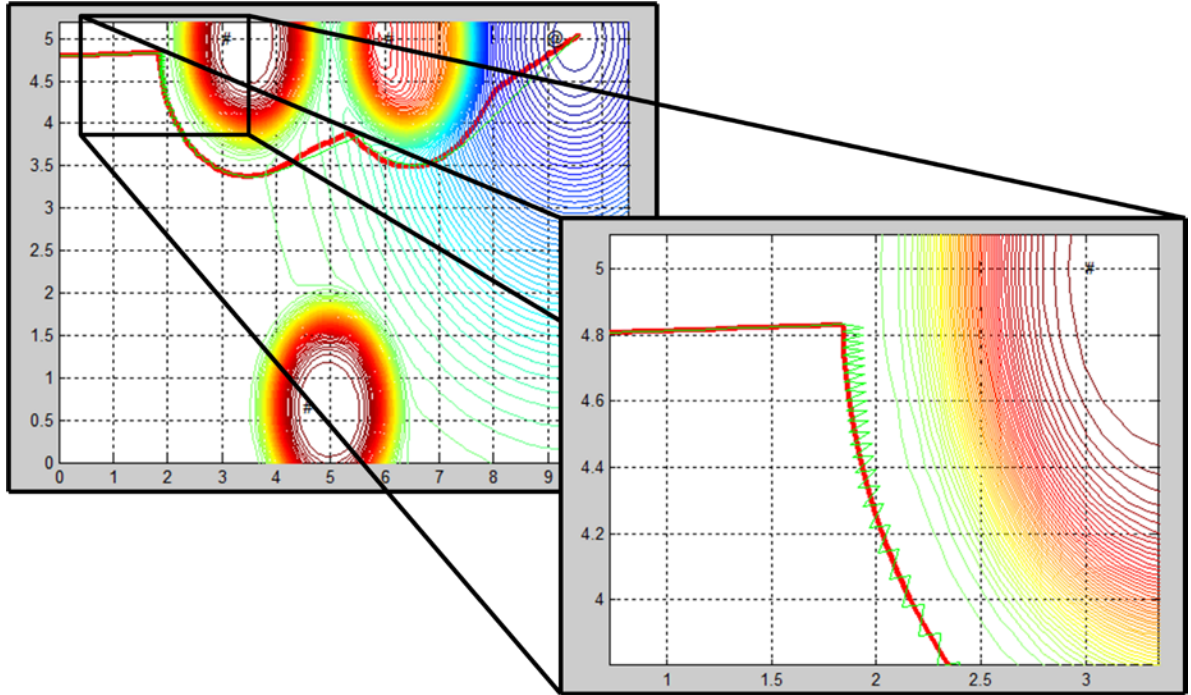


Figure 13 Comparison between gradient descent and newton's method

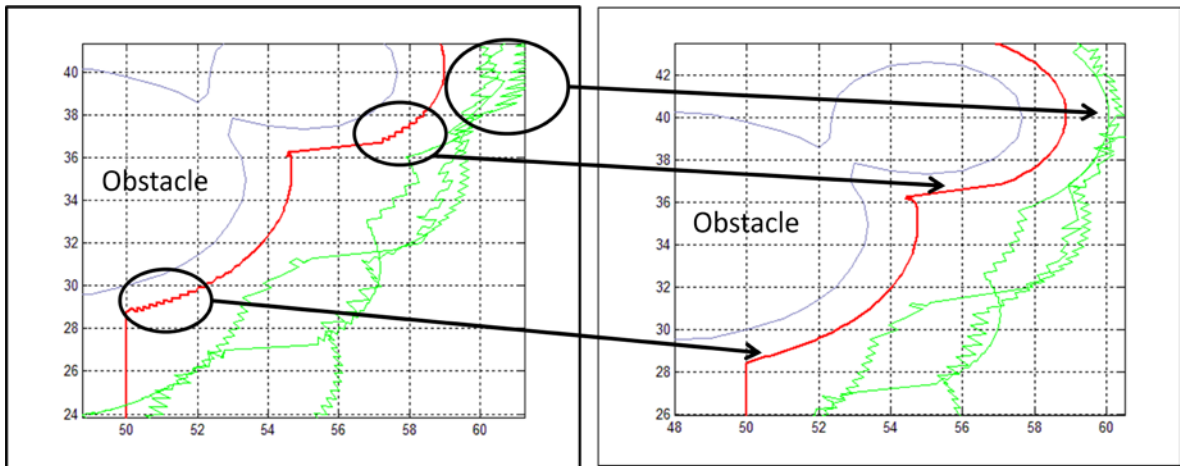


Figure 14 (a)

Figure 14 (b)

Figure 14 (a-b) Comparison between gradient descent and Newton's method for Multi-

Link robot

For multi-link robots, we can apply Newton's method for all the robots in the link. This improves performance, as shown in Figure 14. Figure 14(a) shows performance with the gradient descent method, and 14(b) shows that of Newton's method. Red lines shows the traces of the head and green lines are traces of the tails. The blue line is the boundary of an obstacle. To incorporate Newton's method in to the algorithm we need to add inverse of hessian matrix $B(q)$ in to the control law which gives us a new equation,

$$\dot{q}_i = -\alpha * B(q) * \left(\frac{\partial F_i}{\partial q_i} \right) \quad (\text{Equation 4.2})$$

4.2 Tail Navigation

The body of the snake chases the trace of the head when it's moving. We have used the same concept for our motion planning algorithm. The tail of the first link follows the head of that link by considering it as an attractor, while keeping a safe distance from obstacles, head, and other links. With a known link length and the angle between ($0 \leq \theta \leq 180$) the link heading direction and x-axis at each step we can find the position of the tail easily with respect to the head. The angle of the link with the heading direction after each step is calculated as described below:

As shown in the Figure 15, d_h is the heading direction of head-tail link where orientation of the link is taken as reference direction,

$$d_h = [\cos(\theta), \sin(\theta)] \quad (\text{Equation 4.3})$$

d_h is the direction of negative gradient $-\left(\frac{\partial F_i}{\partial q_i} \right)$ of the navigation function. It has an angle of θ with the x axis so as that $0 \leq \theta \leq 180$.

After each step, the negative gradient direction of tail d_{ng} for the current location of the tail is calculated according to the position of the robot with respect to the obstacles and the target. The negative gradient of the tail has an angle β with d_h such that ($0 \leq \beta \leq 180$).

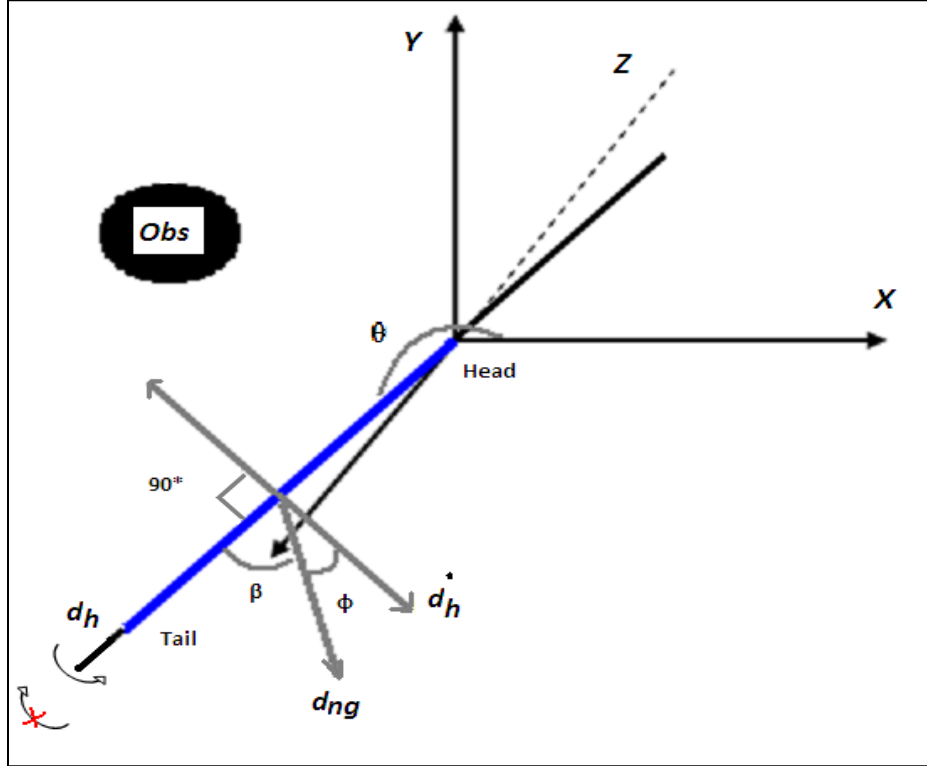


Figure 15 Diagram Shows the Heading Direction of the Link (d_h), Perpendicular Direction (d_h^{\perp}) to the Heading Direction, the Negative Gradient Direction (d_{ng}) of the Tail

4.2.1 Inner Product and Direction of Tail

To avoid collisions with obstacles, the tail of the link should rotate clock-wise or counter-clockwise away from the obstacle. As shown in Figure 16, according to the location of the obstacle, the link will move counter-clockwise. To decide the direction of tail movement, inner product I_p is calculated as:

$$I_p = < d_{ng} \cdot \dot{d}_h > \quad (\text{Equation 4.4})$$

Where \dot{d}_h is in a perpendicular direction to d_h , such that the angle between \dot{d}_h and d_{ng} , φ ranges from $-90 \leq \varphi \leq +90$ rather than the one between d_{ng} and d_h that is $0 \leq \beta \leq 180$. We are taking the inner product because it will give us a positive or negative scalar value according to the relative positions between the obstacles and the head and tail of the robot.

If the value of the inner product is less than zero then the tail will move clockwise to avoid the obstacle. If it is greater or equal to zero then it will move in an anti-clockwise direction. This will give a new value of the angle between the x-axis and heading direction as shown in Equation 4.5. For all the following links, the link ahead will behave as the target.

$$\theta = \begin{cases} \theta - 0.05, & \text{if } (I_p = < d_{ng} \cdot \dot{d}_h >) < 0 \\ \theta + 0.05, & \text{if } (I_p = < d_{ng} \cdot \dot{d}_h >) \geq 0 \end{cases} \quad (\text{Equation 4.5})$$

As shown in Figure 16, gradient of the effect of all forces on the tail will force the tail in the opposite direction of the obstacle. The direction of the tail movement depends upon the gradient of the tail and the heading direction at that instant. The inner product between them will give us an integer value. As shown here, to avoid collision, the obstacle tail should move in an anti clockwise direction by a certain amount that in our simulation we set as 0.5 degree per step. With the value of the inner product calculated from equation (4.4) we can calculate the angle of movement for the link from equation (4.5).

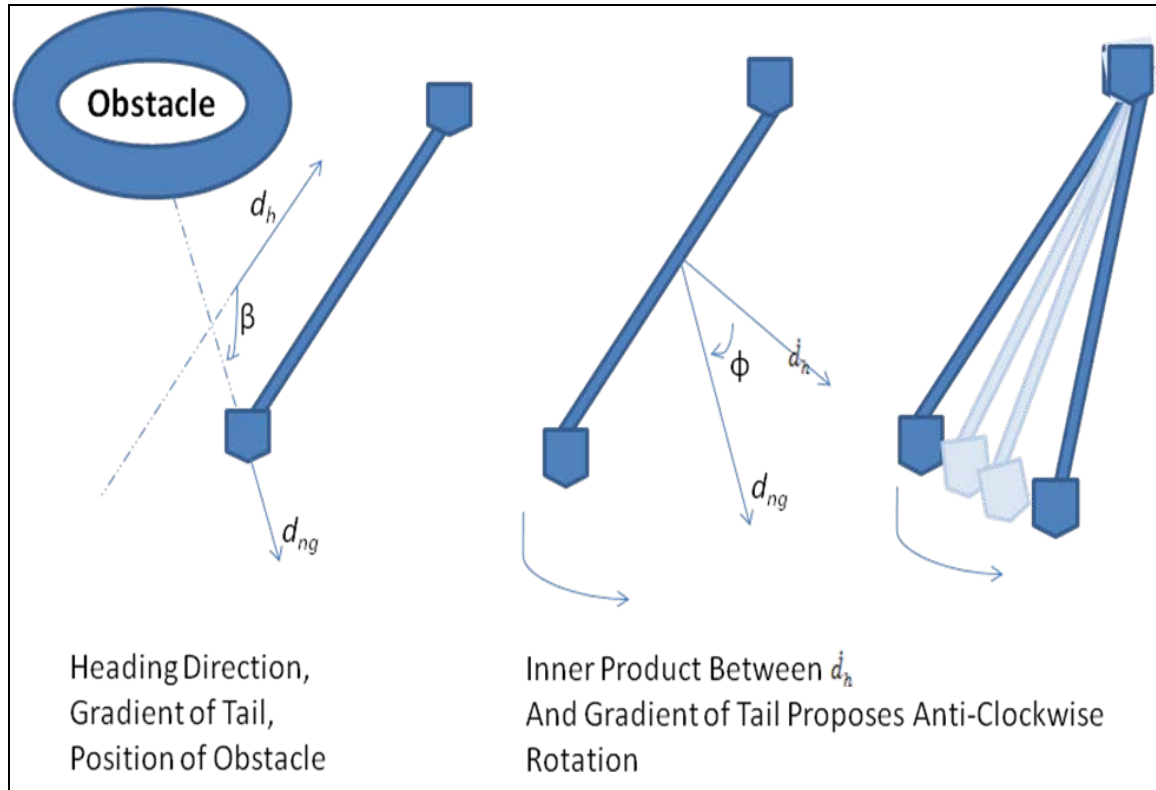


Figure 16 Tail Movements in Accordance with the Inner Product

4.3 Activity Diagrams for the Multi-Link Robot Navigation

4.3.1 Algorithmic motion planning

For each assignment, the first thing to be done is to get and save the details of the work space. For example, the start position, goal position, position of obstacles in the work space. At every step the algorithm will check whether or not the robot reached the goal position and stop the execution of the motion planning algorithm. To reach the goal position, the algorithm will find the position of the next location of the head with help of

hessian and inner product which is explained in Figure 17. The control law is to be transmitted to the robots in terms of speed and turning command.

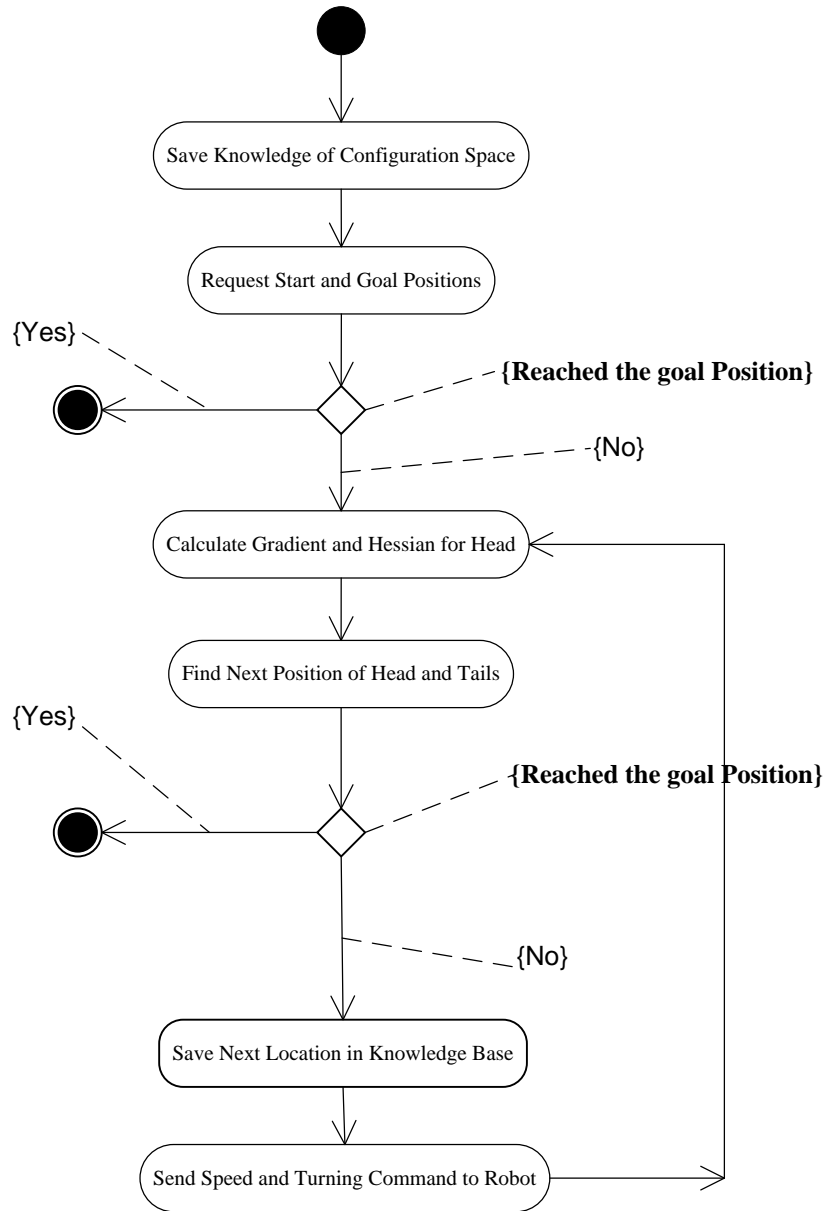


Figure 17 Activity diagram 1

4.3.2 Conditions to be satisfied

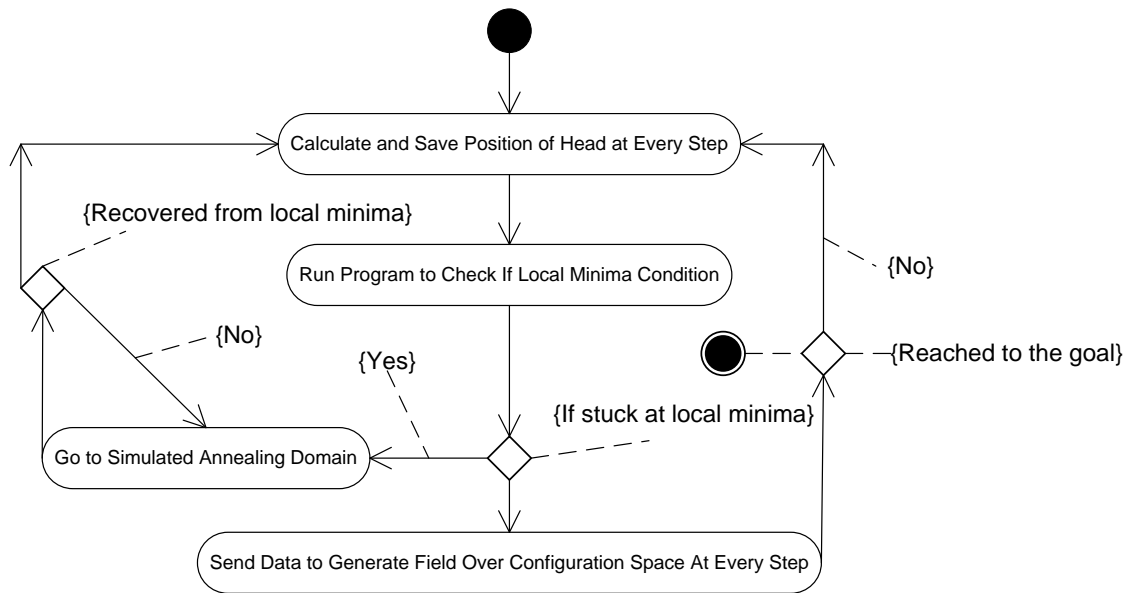


Figure 18 Activity diagram 2

Once the algorithm calculates and saves the position of the Head and Tails at every step it will check whether or not the Link-Robot is trapped in to local minima. In case it is trapped, the control will be transferred to the Simulated Annealing domain. If it is not trapped, the algorithm will send the new locations and knowledge of the work space to generate the potential fields over the configuration field and will check whether or not the robot has reached the goal position. See Figure 18.

4.3.3 Movement during APF Domain

With the location of Head algorithm, the heading direction and the locations for successive tails can be calculated. With knowledge of head, tails, and obstacles, the effect of attractive field and repulsive fields will be calculated. That in turn will be helpful to calculate the hessian with help of synthesis of all the forces. Next, the location of head will be calculated with help of the hessian value. With the heading direction and the

hessian value we can find the inner product which in turn will decide the movement direction for tails to avoid obstacles. With help of the location of head and movement of tails, we can figure out the control law for the robots. The function to find the heading direction and the function to find the effect of attractor and repulsor, do not run in parallel to one another. In Figure 19, the two are shown in parallel to provide a better understanding of the algorithm.

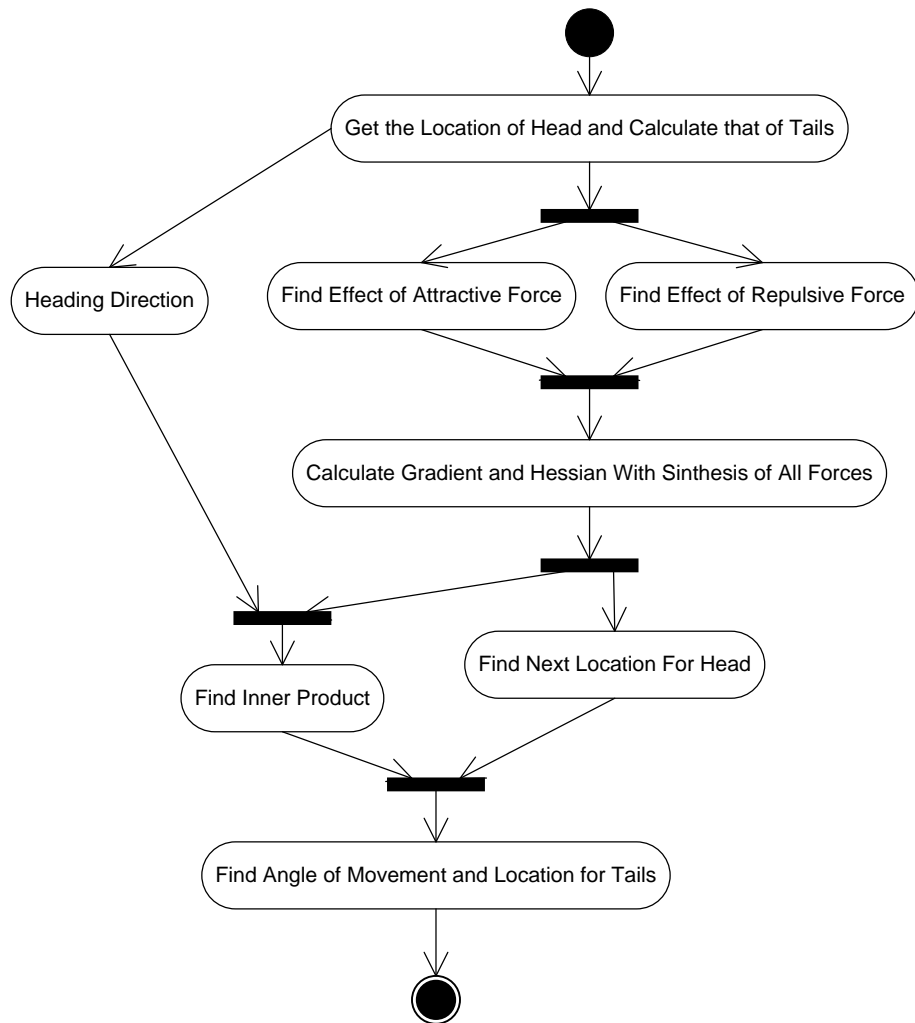


Figure 19 Activity diagram 3

CHAPTER 5

Local Minima Recovery

5.1 Simulated Annealing For Local Minima Recovery

5.1.1 Simulated Annealing

To modify the state, material temperature is used as an adjustable parameter. By controlling temperature the optimum state can be reached. In the physical annealing method, material is heated to impart high energy and then it is cooled slowly. An initial high temperature decreases towards a lower temperature state, and the controlled decrease of temperature results in dissipation of an absolute minimum of energy. Metropolis algorithm causes a small random displacement at every step to make improvement in the actual configuration step by step with a sequence of moves. This sequence of repeated moves constitutes Markov chain where each configuration has a certain set of neighbor configuration. Move towards one of them causes small change in the function.

Optimization methods that use the annealing technique are known as Simulating Annealing (SA) methods. Since its conception, SA is used effectively in various fields such as the design of electronic circuits, collection of household garbage, data-processing and image processing. For motion planning algorithm, SA considers each location of the work space as a physical state of an object under heating and cooling. Optimization

function used in SA is similar to the energy characteristics of annealing. Heating results in randomly distributed higher energy states and slow cooling enables lower energy states to be found. Similarly, optimization with SA locates nearby random solutions with respect to the current state and a navigation function decides probabilistically whether or not to move from the current state to a neighbor state after each step. The parameter temperature is used in the optimization method to control execution of the algorithm, as well as to change the value of acceptance probability after each step.

5.1.2 Parameters for Annealing schedule

Annealing schedule depends upon a temperature parameter which can be adjusted according to equation (5.1). At each step, t , the value of temperature parameter will decrease by a fraction of α_t .

$$T_{(t)} = \alpha_t * T_{(t-1)} \quad (\text{Equation 5.1})$$

Where, $\alpha_t = 0.9$

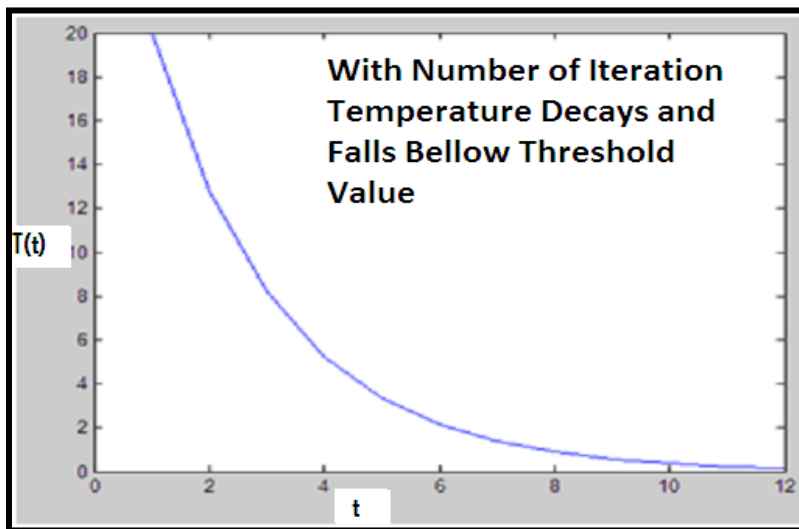


Figure 20 Relation between number of iteration t and decay of temperature

In the simulation we have taken $T(t) = \alpha_t^2 T(t-1)$, where the value of α is taken as 0.85 which results in the curve shown in Figure 20. This characteristic shows that temperature will eventually fall below the threshold temperature value (0.1430) from initial high temperature T_0 which implies that the proposed APF and SA approach will not stay in the SA process state forever and will eventually go back to the APF process state.

If the robot comes out of local minima and reaches a location from where it can move smoothly again by using the APF approach it will do so even if the temperature is over the threshold value.

5.1.3 Acceptance Probability

The probability of making transition to the neighbor configuration from the current configuration is called acceptance probability. It depends upon the value of the navigation function at both the current and neighbor location, and temperature parameter. Value of the navigation function at the neighbor location must be less than 1, which means it should be out of obstacle range. Acceptance probability will allow the uphill movement that prevents the robot to trap at local minimum which is a state worst than global minimum but better than its neighbors. The acceptance rule is:

$$r < e^{\left(-\frac{\Delta F}{T}\right)} \quad (\text{Equation 5.2})$$

Where, $r \in [0,1]$

In the equation above, r is a randomly drawn integer in between 0 and 1, and T is current temperature. Figure 21 shows how the acceptance probability $P(t) = e^{\left(-\frac{\Delta F}{T}\right)}$

decreases towards zero after several iterations when initial temperature T_0 set at 20, threshold at 0.1430, and average ΔF at 0.8. Also, the value of α_t is less than 1 and T_0 is the initial high temperature. If the initial temperature is a high value, the acceptance probability will choose 1, which makes the rate of acceptance of neighbor high in the initial state of the algorithm. This phenomenon allows the acceptance of transition in most cases, but with temperature decreases towards low values, the rate of acceptance decreases. Decrease of temperature causes the acceptance probability to fall rapidly which makes less accepted transition after a several number of iterations. This kind of scheduling will make sure that initially the algorithm will allow degradation of the navigation function value, but after several steps it won't allow uphill moves and once the probability falls below 0 it will allow only downhill moves.

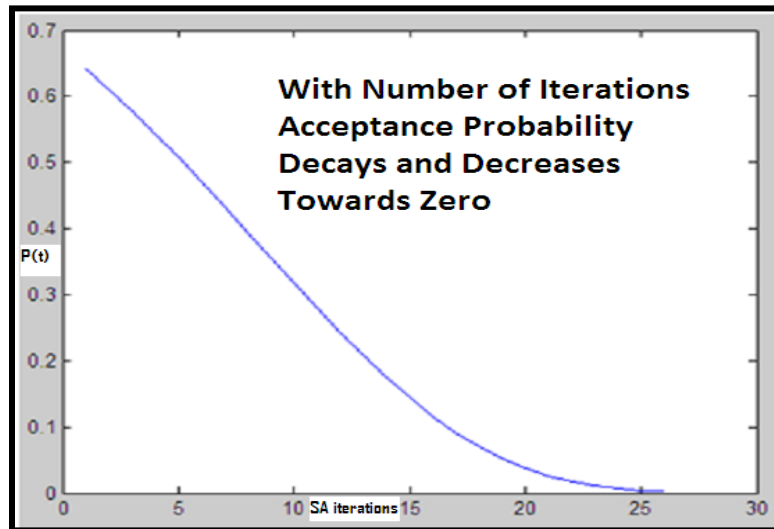


Figure 21 Relation between number of iteration t and acceptance probability $P(t)$

5.2 Modifications in SA

Advantages of SA are that it generally achieves a good quality solution, it's easy to deploy, and is also flexible [47]. Overall, the hybrid control methodology using APF

integrated with SA has the advantage of the APF method in that it helps to recover from local minima.

The SA method has been modified so that it selects a few random neighbors from the vicinity of the robot and then chooses the best candidate amongst them instead of taking a random neighbor. Selection of the best candidate is based upon the value of navigation function at every selected neighbor. With the change, we are still allowing the algorithm to take an uphill move, but making sure that the uphill move is the best amongst all possible moves at each step.

The algorithm also makes sure that it will not choose the formerly visited location during that particular SA process state. This will avoid repeated moves between neighbors and avoid the areas which have already been visited, as shown in Figure 22. Also, the selection of a best neighbor amongst the other neighbors eliminates any backward tracking. This phenomenon helps the algorithm to lower the convergence time. Once the neighbor is selected for transition, to move at that location, the robot will again follow the artificial potential field method for multi link. Once it reaches the neighbor, it will again repeat all the steps in the annealing method.

As shown in Figure 22, point 6 (red spot) shows a local minimum and locations 1 to 5 are some of the randomly selected neighbors. The best neighbor amongst them will be at the point shown as 1, having the least navigation function value. This depends upon the positions of the point with respect to the obstacle and target (shown as @). Therefore, point 1 will be accepted as the local target for the first iteration of the SA process state. After the first step, the algorithm will consider the next set of neighbors and move to the best amongst those and will choose a location from where it already came through during

the time it is in this SA process state. During the next step, while selecting the best neighbor, it won't choose points that are visited once or that are compared with the selected neighbors as shown in the figure.

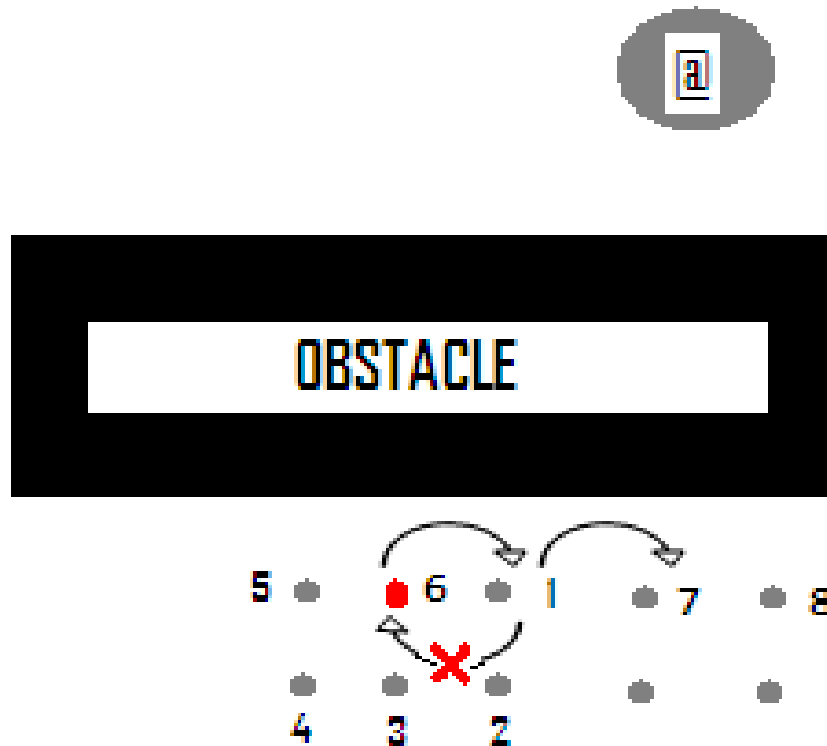


Figure 22 Local minima Recovery by Modified SA

5.2.1 Comparison between two approaches

Table 5.1 shows the comparison between the number of steps to recover from a local minima situation with our algorithm and conventional approach where a random neighbor is selected. The scenario used to compare these two methods is the same as shown in Figure 6.8 simulation results.

	SA	Modified SA
Trial 1	102	39
Trial 2	79	39
Trial 3	65	42
Trial 4	98	42
Trial 5	110	42
Trial 6	108	41
Trial 7	105	38
Trial 8	73	38
Trial 9	92	38
Trial 10	110	42

Table 1 Comparison between SA and modified SA in terms of number of steps to recover from local minima

5.3 Integration of APF and SA to recover from Local Minima

During local minima occurrences the robots get stuck at one location or oscillate around a certain point. By analyzing the average difference between the positions of the robot for 10 consecutive movements, we can decide whether or not it is trapped in a local minimum. In case the algorithm finds out that the robot is trapped in a local minimum it will transfer the control of execution to the simulated annealing mode.

The first step in the SA process state is to adjust the annealing parameters that are rate of acceptance, initial temperature, and fraction by which the temperature decreases at every step. The annealing algorithm will start finding the set of random neighbors once it adjusts the schedule. The algorithm computes random neighbors in a certain range of the robot's current location. By comparing the value of navigation functions at all these random neighbors, it chooses the best neighbor amongst them.

Let us consider a local minimum, point X, and a chosen neighbor Y. $F(X)$ and $F(Y)$ is the corresponding navigation function value for locations X and Y. If the movement between X and Y causes a decrement in the navigation function ($F(Y) \leq F(X)$) then it is accepted unconditionally. In case the movement causes an increase in ΔF (i.e. $F(Y) > F(X)$) then it will be accepted with a probability value of $e^{\left(-\frac{\Delta F}{T}\right)}$. This uphill move causes degradation in the navigation function; in the SA algorithm accept the move and that causes a temperature reduction by the fraction of α as shown in the equation for temperature decay. The robot will move to the location Y, after that the process will be repeated until the robot escapes from the local minima or the threshold temperature, T_h , is reached. In case the neighbor is rejected more than 3 times, it will come out of the SA method again.

During the actual motion, to point Y from X, the control law of the robot is derived by the APF algorithm. Two conditions must be satisfied for a neighbor to be chosen as the next point; the acceptance probability $e^{\left(-\frac{\Delta F}{T}\right)} \geq 0.2$ and $F(Y)$ should be less than 1. The reason behind this is that the value of the navigation function will be equal to or more than 1 inside the region represented by the super-scribed circle of the obstacles' repulsive fields.

Once the robot escapes from a local minimum, it again follows the hessian value of the navigation function to reach the goal. The robot is at most using two modes while moving. In the normal mode it travels with the help of the APF algorithm and in a local minima recovery mode it switches to the SA algorithm.

The steps of the algorithm to recover from local minima are:

```
Step1. Local minimum is set as a starting point for Simulated Annealing mode

Step 2. Selection of Annealing Strategy
{
    Select proper value for Initial Temperature,
    Cooling rate and threshold value for Temperature
}

Step 3. While  $T \geq T_h$  and stuck at local minima,
{
    Search random neighbors of  $X$ 
    Find the best neighbor having least value of  $F(Y)$ 
    Set  $\Delta F = F(Y) - F(X)$ 
    If  $\Delta F \leq 0$ , set  $Y$  as  $X$  as well as the local goal position
    Else set  $Y$  as  $X$  with probability  $P = e^{\left(\frac{-\Delta F}{T}\right)}$ 
    (We have taken a value  $R$  same as our step size and if  $P > R$  then  $Y = X$ )

Step4. To reach the local goal ( $Y$ ) transfer the control to APF mode
    (Send the value of current and next location to APF)
    Next location = current location, then move back to SA

Step5.  $T = \alpha T$ 

Step6. IF  $F(Y) \leq F(X)$ , escaped from local minima
    Return to the APF mode with the current location.
    Else if rejected more than 3 times
Return to APF
}
Return
```

Code 1: Local Minima Recovery

5.4 Activity Diagram of Recovery from Local Minima with Help of Modified SA and APF Integration

To check whether or not the robot is stuck at local minima we refer the average movement of robot for 10 last steps after each step. In case of free movement, it will remain in APF domain. If it is stuck, the control will go over SA domain. First of all, the annealing schedule is set with the parameters temperature and acceptance probability. The annealing schedule algorithm searches for the random set of neighbors located around the current location once the parameters are set. The best one amongst all neighbors is selected by comparing the value of navigation function at all the neighbors. Acceptance probability will decide whether or not to move at that location and in case it won't allow it, the algorithm will find a bunch of random neighbors again.

Once the acceptable neighbor is selected, movement to that location will follow the APF method. The location of the neighbor will be saved in to the knowledge base and temperature parameter is adjusted. If the robot has recovered from local minima or the value of temperature is below threshold, control will go back to the APF domain. In case the current location still has less value of navigation function compared to the one at local minima and the temperature is still at a higher value, than the algorithm will find a next set of random neighbors and go through all the steps again.

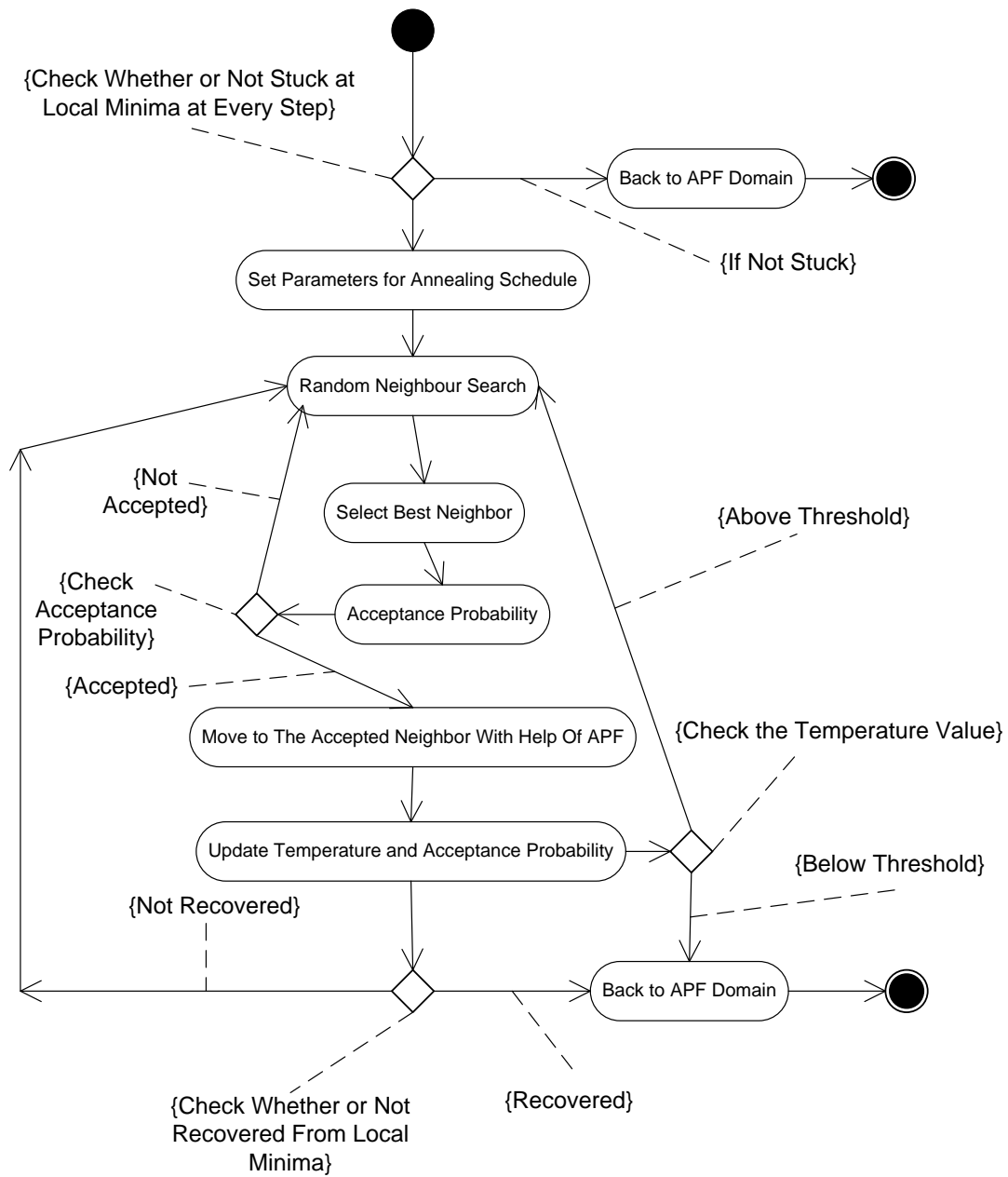


Figure 23 Activity diagram 4

CHAPTER 6

Implementation and Simulation Results

6.1 Implementation of Algorithm

6.1.1 Use Case Diagram

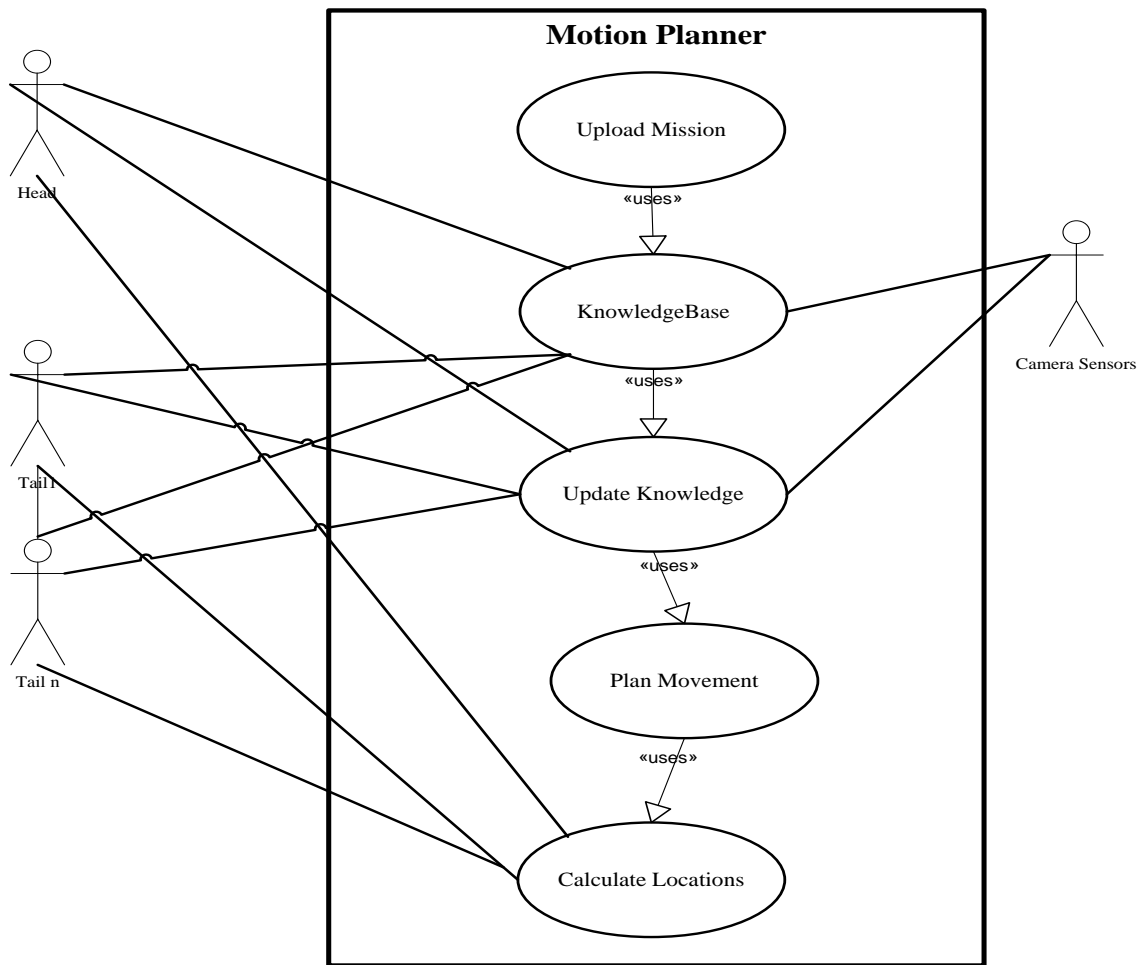


Figure 24 Use Case Diagram of Motion Planner

Figure 24 shows the overall implementation of the motion planning system with use case diagram. Robots considered as a head and tails are actors that communicate with the centralized motion planner. Sensors provide the planner information about the work space which is stored in to the knowledge base. Once the assignment uploaded in to the knowledge base, motion planner will start executing the algorithm to connect the start configuration with the end configuration. At each step the algorithm will calculate the next location, update the knowledge base with that location and send the location to the robots in the link as well.

6.1.2.1 Overview of Motion Planner

Figure 25 Message Diagram: Overview of Motion Planning System.

Figure 25 shows the message diagram for the motion planning system, which illustrates the sequence of the function calls. The left side of the diagram describes the sequence of calls between the robots and the communication channel between the robots and the centralized motion planner.

Sensors mounted on robots may be a GPS system that gives the current location of the robot to the controller inside the robot module. Just after the assignment is uploaded, the motion planner will ask the robot for the initial location. With the work space information, target, and start location in the knowledge base, the motion planner will plan movement and find the next position for robots. Plan_movement() function call will contains the information about work space, start and goal configuration while the function called Find_position() will send the information regarding the movement in terms of the angle of movement and step size to the position calculator that in turn will find the next location and will update the knowledge base. Control law, derived from motion planner, is used as a speed and turning command for the robot controller.

6.1.2.2 Overview of the Algorithm

Figure 26 explains the sequence of message calls in between knowledge base, position calculator, potential field generator function and the function that decides whether or not the robot is trapped in to local minima. Once the Set_Position() call updates the knowledge base with the data of the next location and direction of movement, knowledge base will send all the detailed information to the function which generates the potential field over the work space. The function call Check_LM() will contain the details of the last ten steps of the robot, with the help of the Local_Minima_Detector function

which will decide whether or not the robot is trapped in to Local Minima and gives the feedback. Check_Target() call contains details of the current location which is compared with the location that the GPS gives to decide whether or not the goal location is reached.

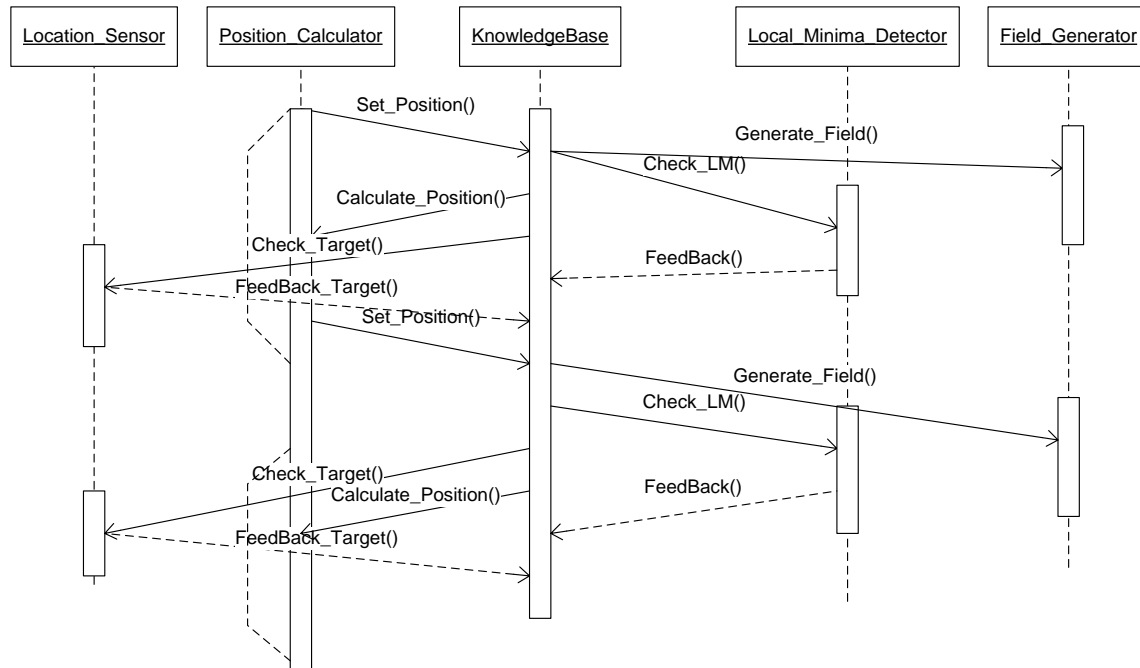


Figure 26 Message Diagram: Algorithm Overview

6.1.2.3 Overview of Artificial Potential Field Method for Motion Planning

Knowledge base will send the information with a function called Calculate_Force() to the function that will generate potential field with Gaussian functions. Find_Hessian() call will contain details of the attractor field and repulsive field in terms of navigation function, which trigger the function Hessian_Calculator to calculate the value of hessian for the navigation function. The InnerProduct_Calculator function will calculate the inner product in between the gradient of the tail and the heading direction. The Find_IP() function call consists of data regarding to gradient and

heading direction. With the help of direction of the movement and the hessian value movement calculator the next location will be calculated and will update the knowledge base.

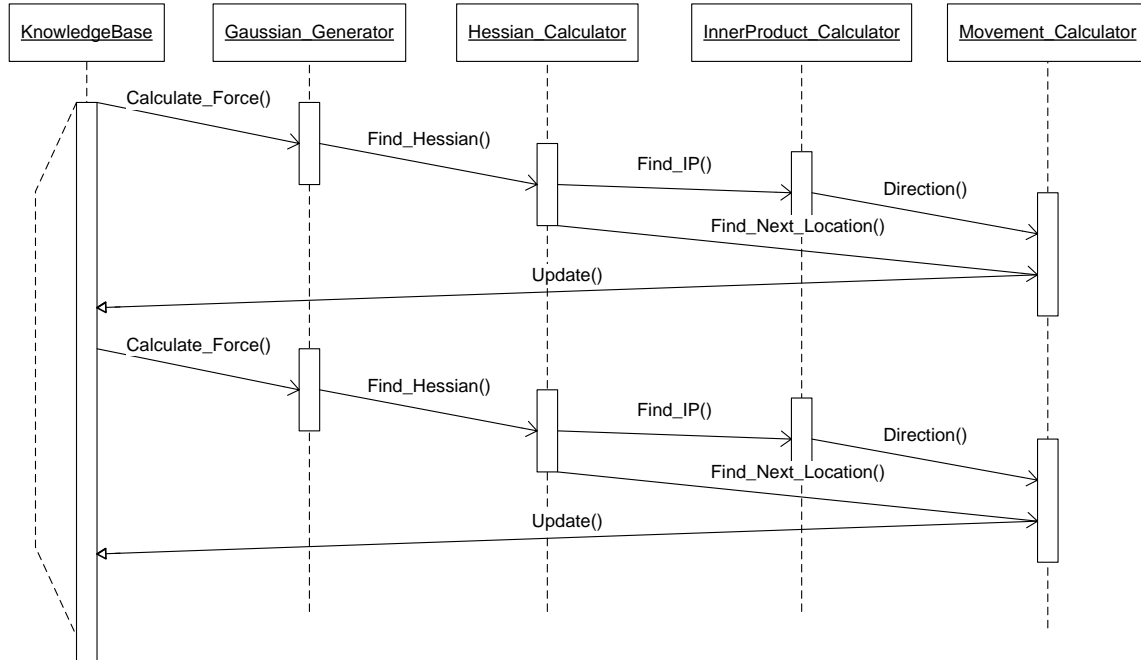


Figure 27 Message Diagram: APF Method

6.1.2.4 Local Minima Recovery

At every step, when knowledge base is updated with the value of the next location, it will generate a function called `Check_LM()` which contains details of the ten previous locations. The Local Minima detector function will analyze the data and will decide whether or not the robot is stuck. It will thereafter accordingly send a feedback message. In case of a trap situation, it will generate a function called `Set_parameters()` for the Simulated Annealing scheduler. This sets the value of temperature, threshold value of temperature, and acceptance probability. After adjusting the schedule for annealing, a

function called Find_Nighbor(), containing the value of current location, triggers the function Neighbor_Locator which in turn randomly searches for a set of neighbors and then finds the best one according to the value of navigation function. The Decision_Maker function will decide whether or not to accept the selected neighbor based on acceptance probability and gives corresponding feedback. In case of accepted neighbor the location is forwarded to the Motion Planner which generates the control law for that move. The terminator will decide whether or not the threshold value for temperature is reached, or the robot has recovered from the local minima and gives feedback to the motion planner. It simultaneously sends the current location to the knowledge base to update the array of data regarding the locations of the robot at each step.

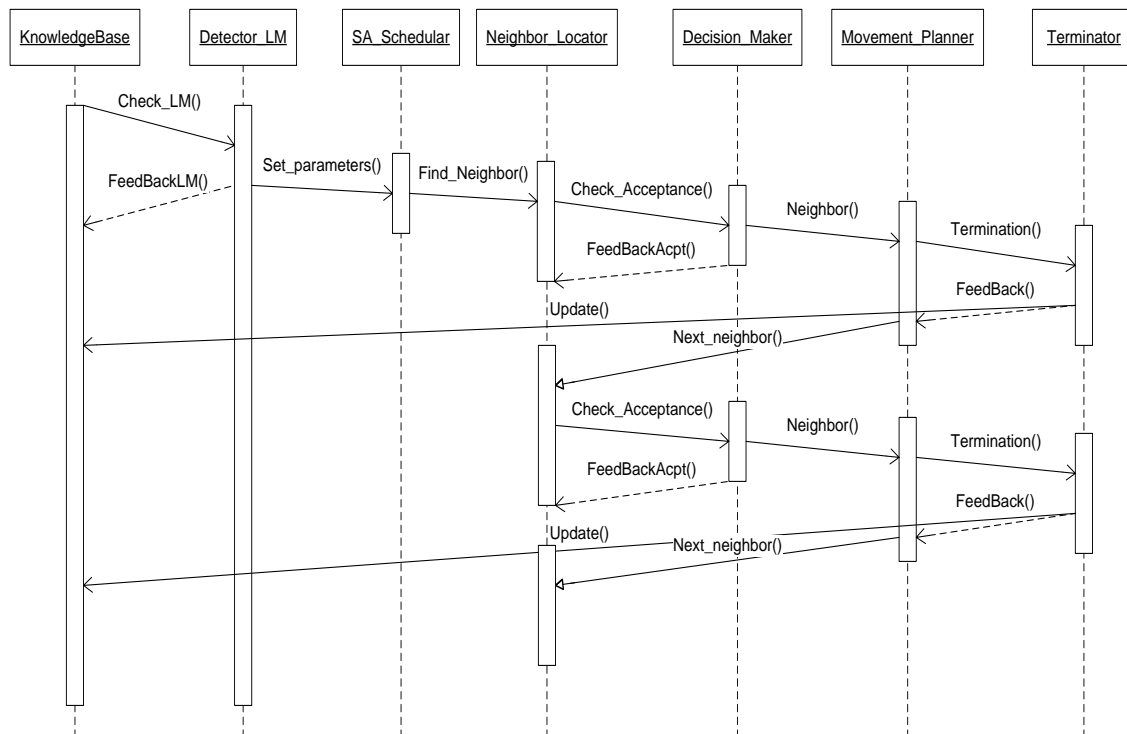


Figure 28 Message Diagram: Local Minima Recovery

6.2 Simulation Scenarios

To test the capabilities of algorithm in different situations, three different kinds of scenarios have been created. The first scenario displays multi-link robot navigation in a work space like a narrow corridor, sewer system or pipeline. To create such surroundings the obstacles are placed close to each other and the link robot is given an objective to go through the corridor which has a couple of openings.

The second scenario shows how the robot recovers from local minima with the help of Simulated Annealing in a workspace containing sparsely spaced obstacles. It seems as an outside environment that has obstacles away from each other in a large opening with a couple of locations where local minima can occur.

The third scenario displays the co-ordination behavior between two multi-link robots when they reach an opening simultaneously. This was created to test the capability of algorithm to handle multiple multi-link robots simultaneously in an objective that needs co-ordination between them.

6.2.1 Performance of the Algorithm in between Closely Spaced Obstacles

To measure the capability of the algorithm to navigate a multi-link robot through closely spaced obstacles, we performed a simulation in an environment like a pipeline or sewer system as shown in Figure 29. In this figure, the multi-link robot is shown as a blue line. The *R number* printed close to the head of the link shows the instantaneous step number of the robot movement. The red line shows the trace of the head movement and

the green lines are the traces of the successive tails. This labeling convention is used throughout all the results of the different scenarios displayed from Figures 29 to 34.

The multi-link robot in Figure 29 has 3 links, as shown, and reaches the goal, shown by the @ symbol, without getting stuck as well as avoiding collisions with the objects in the environment and avoiding the openings.

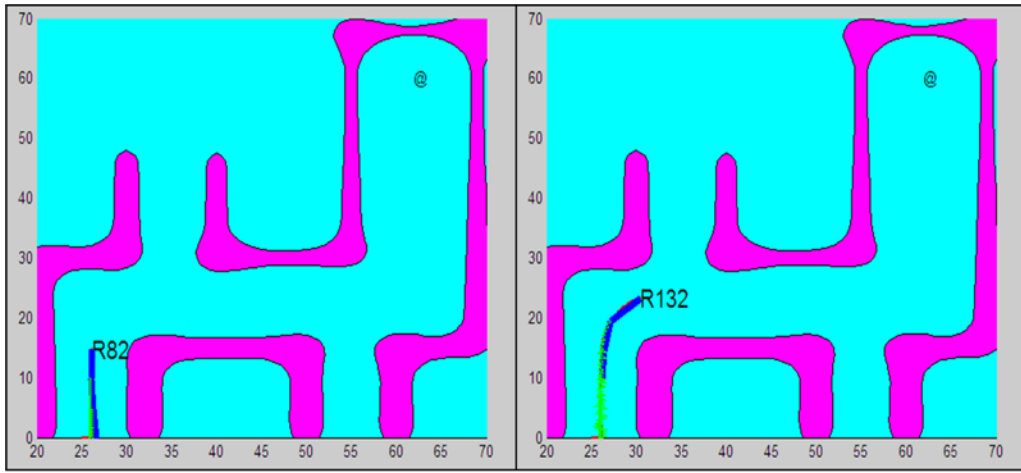


Figure 29 (a)

(b)

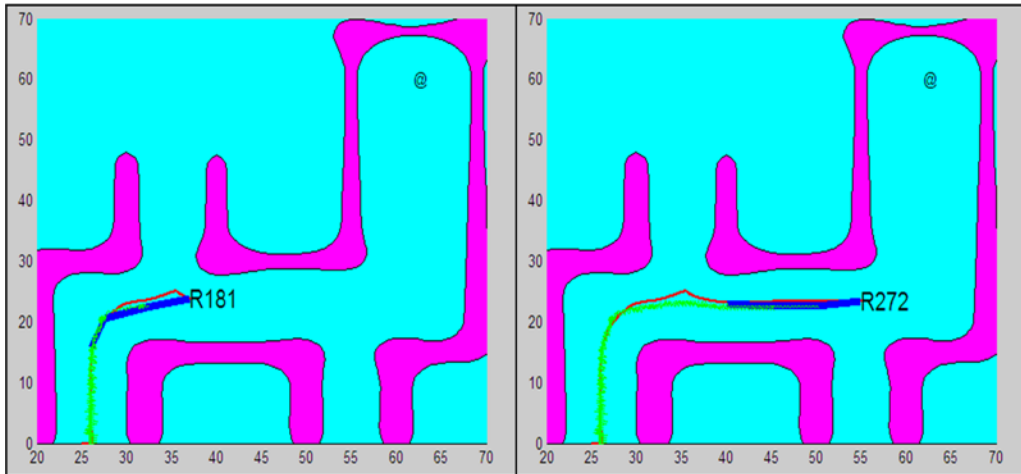


Figure 29(c)

(d)

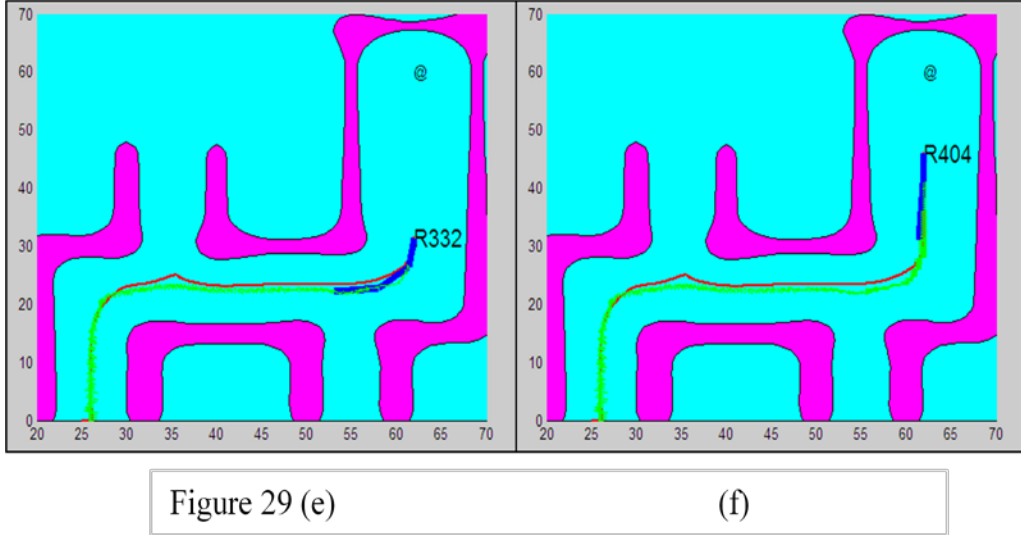


Figure 29 (a-f) Simulation for 3 Link Robots in Narrow Passage

6.2.2 Local Minima Recovery

To analyze the capability of the combined APF and SA algorithms to help the robot recover from a local minimum, a simulation workspace consisting of a large outside area having sparsely placed obstacles has been used as represented in Figures 30 and 31.

The workspace has 4 convex obstacle shapes, each represented by a superscribed oval. The possible locations where local minima can affect the robot motion in this scenario are shown in Figure 30(a) with L. Figure 30(a) shows the robot getting closer to the obstacle and Figure 30(b) shows that without using SA the robot got stuck at a local minima. As shown, with the step of movement the robot is stuck at the location forever.

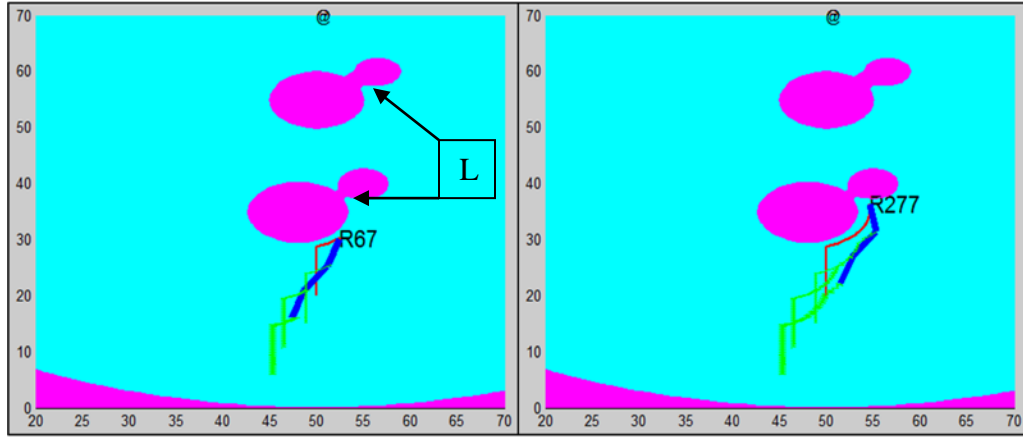


Figure 30(a)

(b)

Figure 30(a-b) Local Minima Trap Without SA

Figure 31 shows that by utilising the APF and SA algorithms the multi link robot reaches its goal with out significant oscillation while avoiding obstacles. Figures 31(a-c) shows the robot is moving to a selected neighbor using the modofied SA algorithm. Once it recovers from local minima the robot again leverages the APF algorithm and successfully navigates it further towards the objective as shown in Figure 31 (d).

In this scenario, the robot recovered from Local Minima in a total of 38-42 steps in 10 different trials. As a comparison, for the same scenario we applied the SA algorithm, but selected the neighbors randomly with use of a conventional SA algorithm. For 10 different trials the robot took 65 to 110 steps to recover from the same location as shown in Figure 32. In the figure vertical axis shows the number of iterations/steps and horizontal axis shows the number of trial.

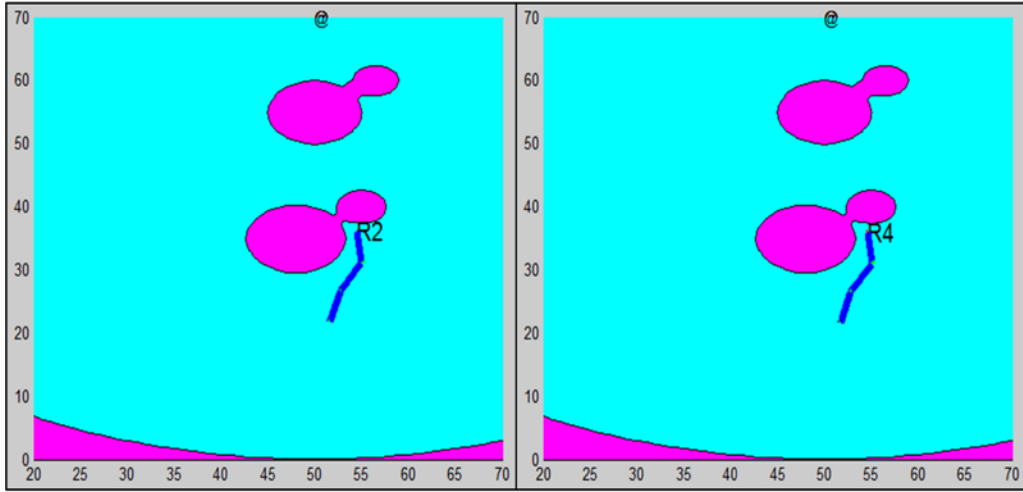


Figure 31(a)

(b)

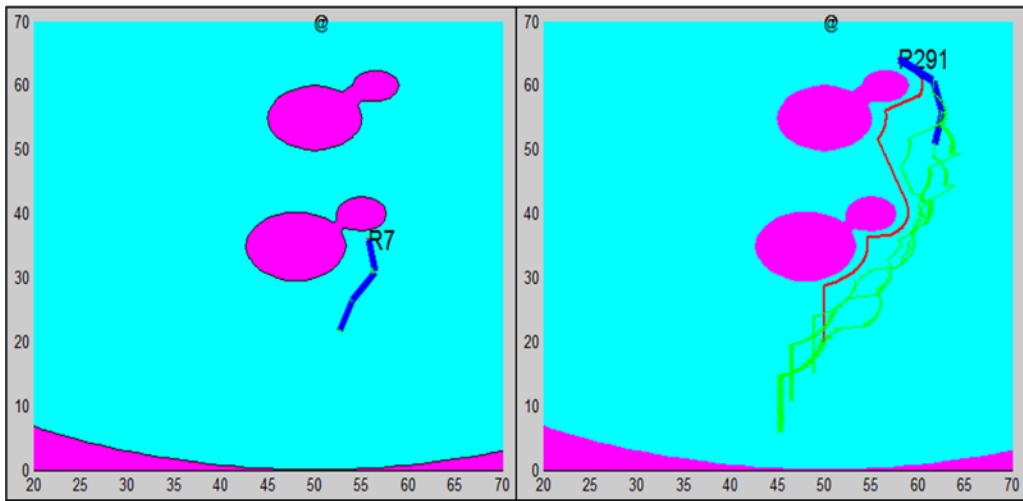


Figure 31(c)

(d)

Figure 31 (a-c) Local Minima Recovery with Modified SA

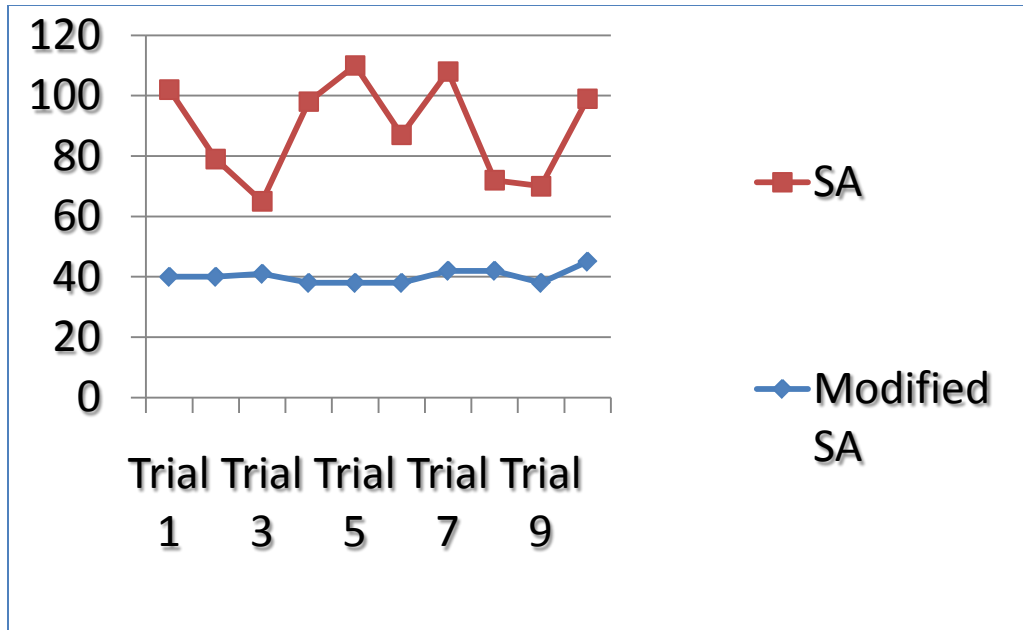


Figure 32 Comparisons between SA and Modified SA Performance

6.2.3 Multiple Robots – Co-operation

Applications like a rescue operation or military operation make it essential to have multiple link mobile robots. In industrial applications, the role of co-ordination between two robots or link-robots is essential to create a multi-agent scenario. The algorithm discussed in this paper successfully converges in multiple multi-link robot scenarios where co-ordination is essential.

To create the team scenario, both the robots should avoid colliding with each other. For this scenario, the algorithm considered the links of the robots as small obstacles. In this manner, a link of a robot will behave as an obstacle for the other. The robot will have a repulsive effect on its fellow robots working in the same objective, here approaching the same opening simultaneously.

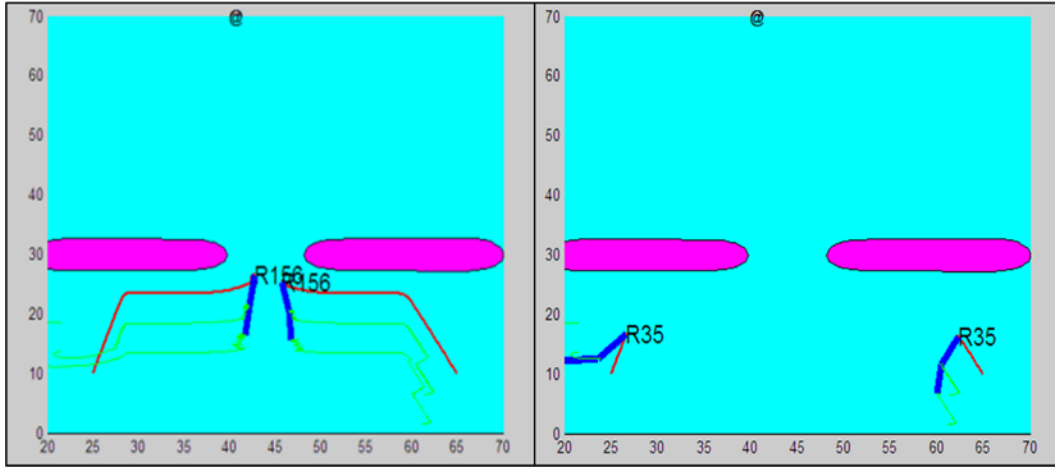


Figure 33(a)

(b)

As shown in Figure 33(a-f), both multi-link robots reach the target while avoiding a collision with obstacles, as well as with each other. Figure 33(a) shows both multiple link robots starting to move towards the goal, reaching the opening simultaneously where collision is possible as shown in Figure 33(b). Instead of colliding, one of them gives safe passage to the other one, waiting for another one to pass by while keeping safe distance and then resuming movement towards the goal, as illustrated in Figure 33(c-e)).

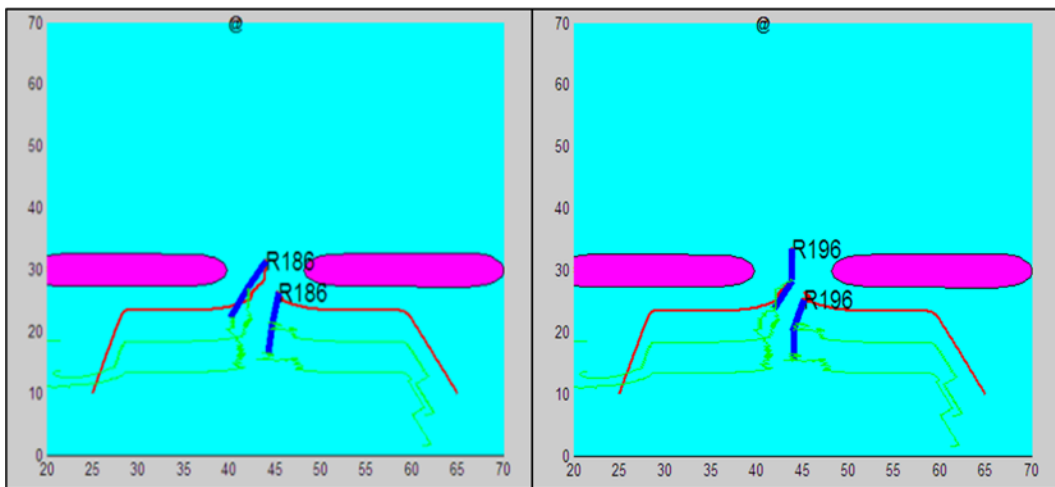


Figure 33(c)

(d)

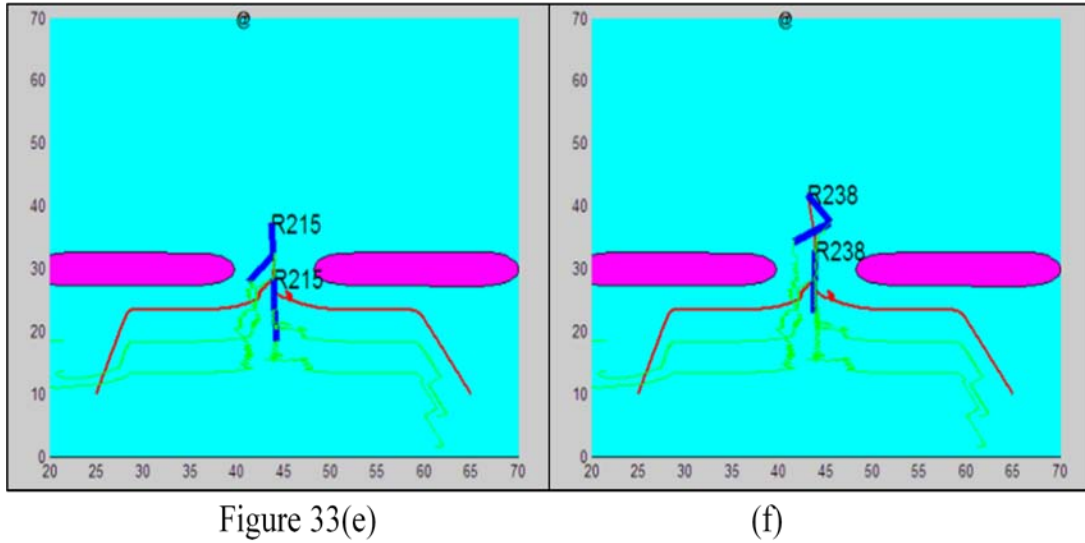


Figure 33(a-f) Co-ordination between Two Multi-links Robots

6.3 Rescue Work Scenario

As discussed earlier, the algorithm can easily be modified so that it can be used for rescue work. As shown in the Figure 34 below, R shows the location where we need to separate one of the link robots from the link having the transmitter device. The 4 robots are creating 3 links and once the last robot is separated the remaining two links will go forward towards the next target, while the separated robot will stay at that location. The green lines show the trace of tails. Initially, there are 3 lines for 3 tails and after location R only two lines for the remaining two tail robots are visible. A blue spot near location R is the separated robot.

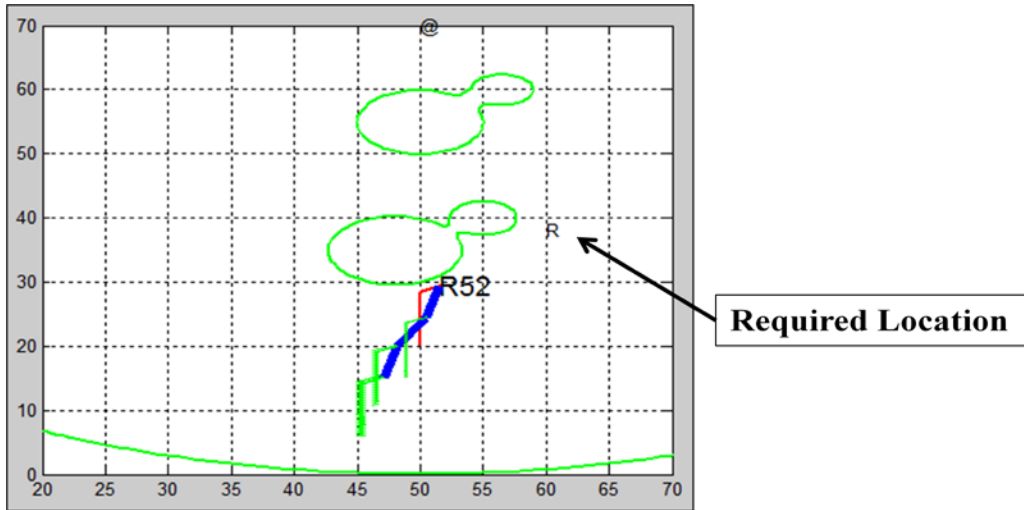


Figure 34(a)

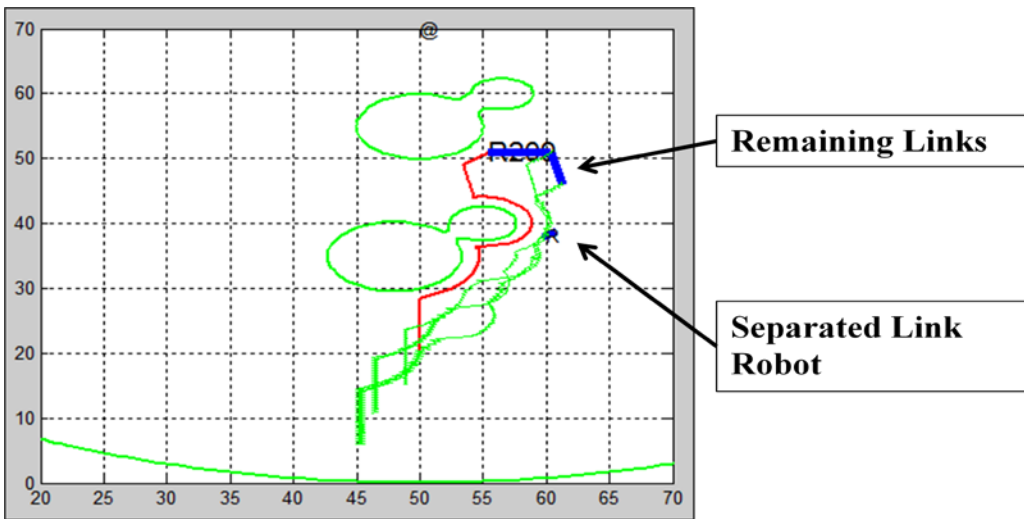


Figure 34(b)

Figure 34(a-b) Rescue Operation Scenarios

Reconfiguration of a multi-agent robotic system asks for higher efforts when implemented in the real world. The simulated scenario used here is just to show the capability of algorithm to work in a rescue operation.

CHAPTER 7

CONCLUSIONS AND FURTHER WORK

7.1 Conclusions

We discussed an algorithm for stable motion planning for a team of multilink mobile robots in the thesis. Simulation results show that:

1. The proposed algorithm converges in such a manner that a multi link robot can move easily in narrow corridors and reach a target
2. Integration of the Artificial Potential Field method with a modified Simulated Annealing algorithm can be used to successfully recover from local minima occurrences
3. The algorithm can also be used for motion planning with multiple multi-link robots working in coordination
4. With modifications, the algorithm converges quicker and with replacement of the Gradient Descent method with Newton's method it works satisfactory in proximity of obstacles
5. The algorithm can be used for different applications such as a rescue task for motion planning or a system where robots moves in formation with modifications

7.2 Possible Future Work

We can use mid-sized mobile robots, such as Koala robots, to implement the algorithm in a real world for different applications. With a properly designed robot we can use the algorithm for WSN (Wireless Sensor Networks), rescue tasks, military applications, as well as industrial applications. The algorithm can be enhanced in a way that the link can be re-configured in different ways. As an example a part of a link can be separated from whole robots, can be rejoined in between, or in case the multi link robot is stuck at a position where the head can move forward, the link can be reversed where the tail will behave like a head. Reconfiguration leads to so many different applications where robots can work as a team and form different shapes for different applications and purposes [35].

Simultaneous Localization and Mapping (SLAM) is a technique that binds together the processes of mapping the work space and robot localization of robot in to that map [48]. With localization we can integrate the information from the sensors and give pass them as an input to our motion planner. The motion planner can accordingly generate the control law and the robot will return the location feedback based on that localization method. In short by integrating our motion planner and SLAM we can generate a complete motion planning application for different purposes.

We can add the effect of moving obstacles, in a protective range of the mobile robots, into the control law as suggested in [33] which allows the robots to move in a direction away from moving obstacles. The change in control law will help the algorithm to operate in an uncertain environment containing dynamic objects.

REFERENCE

- [1] Chee-Keng Yap, “Algorithmic Motion Planning”, *Eds. Hillsdale, NJ: Lawrence Erlbaum*, Vol. 1, 1987, pp. 95-143.
- [2] Dennis Nieuwenhuisen and Mark Overmars, “Motion Planning for Camera Movements in Virtual Environments”, *Utrecht University: Information and Computing Science, Utrecht, The Netherlands*, January 2003.
- [3] Guang Song and Nancy Amato, “Using Motion Planning to Study Protein Folding Pathways,” *Proceedings of the Fifth Annual International Conference on Computational Biology, ACM Press*, 2001, pp. 287-296
- [4] Sujay Sundaram, Ian Remmler and Nancy Amato, “Disassembly Sequencing Using a Motion Planning Approach,” *Proceedings IEEE International Conference on Robotics and Automation, Seoul, Korea*, May 2001, pp. 1475-1480.
- [5] Maciej Kalisiak and Michiel Van de Panne, “A Grasp-based Motion Planning Algorithm for Character Animation”, *The Journal of Visualization and Computer Animation*, 2001, Vol. 12, pp. 117-129
- [6] Koichi Osuka and Hiroshi Kitajima, “Development of Mobile Inspection Robot for Rescue Activities: MOIRA”, *Proceedings Of IEEE International Conference of Intelligent Robots and Systems*, October 2003, pp. 3373-3377
- [7] Shigeo Hirose and Edwardo Fukushima, “Snakes and Strings: New Robotic Components for Rescue Operation”, *Experimental Robotics VIII, Springer-Verlag Berlin Heidelberg*, 2003, Star 5, pp. 48-61.

- [8] Amit Pamecha, Imme Ebert-Uphoff and Gregory S. Chirikjian, "Useful Metrics for Modular Robot Motion Planning," *IEEE Transactions On Robotics and Automation*, v.13, no.4, pp.531-545, 1997
- [9] Benjamin Brown, Jr., Michael Vande Weghe, Curt A. Bererton, and Pradeep K. Khosla, "Millibot Trains For Enhanced Mobility", *IEEE/ASME Transactions On Mechatronics*, v.7, no.4, pp. 452-461, December 2002
- [10] Khatib Oussama, "Real-Time obstacle avoidance for manipulators and Mobile Robots", *The International Journal of Robotics Research*, Vol. 5, No. 1, pp. 90-99, Spring 1986.
- [11] Jean Claude Latombe, "Robot motion planning". Kluwer academic publishers, 1991
- [12] Anthony Stentz, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments", *International Journal of Robotics and Automation*, Vol.10, 1995
- [13] Gerry Dozier, Shaun McCullough, Abdollah Homaifar, Eddie Tunstel, and Loretta Moore, "Multiobjective Evolutionary Path Planning Via Fuzzy Tournament Selection", *Proceedings of IEEE World Congress on Computational Intelligence*, 1998, pp. 684-679
- [14] Byoung-Tak Zhang and Sung-Hoon Kim, "An Evolutionary Method for Active Learning of Mobile Robot Path Planning", *Proceedings Of IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'1997*, pp. 312-317
- [15] Michael Gerke, "Genetic Path Planning for Mobile Robots", *Proceedings Of the American Control Conference, San Diego, California*, June 1999, pp. 2424-2429

- [16] Norio Baba and Naoyuki Kubota, "Collision Avoidance Planning of a Robot Manipulator by Using Genetic Algorithm. A Consideration for the Problem in Which Moving Obstacles and/or Several Robots are Included in the workspace", *Proceedings Of the first IEEE World Conference on Computational Intelligence*, Vol.2, 1994, pp 714-719
- [17] A. S. Rana and A. M. S. Zalzal, "An Evolutionary Algorithm for Collision Free Motion Planning of Multi-arm Robots", *Proceedings Of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1995, pp. 123-130
- [18] Michael Zeller, Rajeev Sharma and Klaus Schulten, "Motion Planning of A Pneumatic Robot Using A Neural Network", *IEEE Control System Magazine*, Vol. 17, June 1997, pp. 89-98
- [19] Jing Yuan, "Collision Identification between Convex Polyhedra Using Neural Networks", *IEEE Transaction on neural networks*, Vol. 6, Nov. 1995, pp. 1411-1419
- [20] Somchai Boonphoapichart, Satoshi Komada and Takamasa Hori, "Robot's Motion Decision-making System in Unknown Environment and Its Application to a Mobile Robot", *IEEE International Conference on Industrial Technology*, Vol. 1, Dec. 2002, pp. 18-23
- [21] Ren Luo and Tse Min Chen, "Development of a Multibehavior-base Mobile Robot for Remote Supervisory Control through the Internet", *IEEE Transaction On Mechatronics*, Vol 5, Dec. 2000, No. 4
- [22] Sadao Akishita, Sadao Kawamura and Kei-ichi Hayashi, "New Navigation Function Utilizing Hydrodynamic Potential for Mobile Robot", *Proceedings Of the IEEE*

International Workshop on Intelligent Motion Control, Bogazici, University, Istanbul, Turkey, 1990, pp. 413-417

- [23] Jarome Baraquand, Lydia Kavraki, Jean-Cloud Latombe, Rajeev Motwani, Tsai-Yen Li and Prabhakar Raghvan, "A Random Sampling Scheme for Path Planning", *The International Journal of Robotic Research*, Vol. 16, No. 6, December 1997, pp. 759-774
- [24] Frank Lingelbach, "Path Planning Using Probabilistic Cell Decomposition", *Proceedings Of IEEE International Conference on Robotics and Automation*, April 2004, pp. 467-472
- [25] Lydia E. Kavraki, Petr Svetska, Jean Cloud-Latombe and Mark H Overmars, "Probabilistic Road Maps for Path Planning in High Dimensional Configuration Spaces," *IEEE Transaction On Robotics and Automation*, Vol. 12, 1996, pp. 566-580
- [26] Edward S. Plumer, "Neural Network Structure for Navigation Using Potential Fields," *Proceedings International Joint Conference Neural Networks*, Vol.1, 1992, pp. 327-332
- [27] Kun Hsiang Wu, Chin Hsing Chen and Jiann Der Lee, "Genetic-based Adaptive Fuzzy Controller for Robot Path Planning", *Proceedings Of the Fifth IEEE International Conference on Fuzzy Systems*, Vol. 3, 1996, pp. 1687-1692
- [28] Elon Rimon and Daniel E. Koditschek, "Exact Robot Navigation Using Artificial Potencial Functions", *IEEE Transaction on Robotics and Automation*, Vol. 8, No. 5, October 1992

- [29] Richard Volpe and Pradeep Khosla, "Manipulator Control with Superquadric Artificial Potential Functions: Theory and Experiments", *IEEE Transaction on Systems, Man, And Cybernetics*, Vol. 20, No. 6, November/December 1990
- [30] Jin-Oh Kim and Pradeep K. Khosla, "Real-Time Obstacle Avoidance Using Harmonic Potential Functions", *IEEE Transaction on Robotics and Automation*, Vol. 8, No. 3, June 1992
- [31] Shuzhi Sam Ge and Yan Juan Cui, "New Potential functions for mobile robot path planning", *IEEE transaction on robotics and automation*, Vol. 16, No. 5, October 2000
- [32] Leng Feng Lee, "Decentralized Motion Planning Within An Artificial Potential Frameworks (APF) For Cooperative Payload Transport By Multi-Robot Collectives" *Masters of Science Dissertation*, The State University of New-York at Buffalo. December 2004. [<http://www.eng.buffalo.edu/~llee3/Research.html>]
- [33] Dr. Jing Ren, "Potential Field Based Motion Planning for Search and Forage Task", *Masters of Engineering Science Dissertation*, University of Western Ontario. 2003
- [34] H. M. Ha, A. D. Nguyen and Q. P. Ha, "Controlling Formation of Multiple Mobile Robots with Inter-Robot Collision Avoidance", *Proceedings Of Australian Conference on Robotics and Automation, (ACRA 2005), Sydney, Australia*, December 2005
- [35] *University of the West of England/ press note- UWE investigates evolving 'swarm' robots*, Issue date: 13/03/2008
[\[http://info.uwe.ac.uk/news/uwenews/article.asp?item=1231\]](http://info.uwe.ac.uk/news/uwenews/article.asp?item=1231)

- [36] Shigeo Hirose, Takaya Shirasu and Fumihiko Fukushima, “A Proposal for Cooperative Robot “Gunryu” Composed of Autonomous Segments”, *Proceedings Of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, Munich, Germany*, Sept 1994, Vol.3, pp. 1532-1538
- [37] Kenneth A. McIssac, Jing Ren and Xishi Huang, “Modified Newton’s Method Applied to Potential Field Navigation”, *Proceedings Of the 42nd IEEE Conference on Decision and Control Maui, Hawaii USA*, December 2003
- [38] Yoram Koren and Johann Borenstein, “Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation”, *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* Sacramento, California – April 1991
- [39] Prahlad Vadakkepat, Kay Chen Tan and Wang Ming-Liang, “Evolutionary artificial potential fields and their application in real time robot path planning”, The National University of Singapore
- [40] Scott Kirkpatrick, Dan Gelatt Jr., Mario Vecchi, "Optimization by Simulated Annealing", *Science*, 220, 4598, 671-680, 1983
- [41] Dimitris Bertsimas and John Tsitsklis, “Simulated Annealing”, *Statistical Science*, Vol 8, No.1 page 10-15, 1993
- [42] Johann Dreo, Alain Petrowski, Patrick Siarry and Eric Taillard, “Metaheuristics for Hard Optimization”, *Springer-Verlag, Berlin Heidelberg* 2006
- [43] Farrokh Janabi-Sharifi and D Vinke, “Integration of the Artificial Potential Field Approach with Simulated Annealing for Robot Path Planning”, *Proceedings of the IEEE International Symposium on Intelligent Control, Chicago*, pp. 536-541, 1993

- [44] Yong Hwang and Narendra Ahuja, “Gross Motion Planning – A Survey”, *ACM Computing Surveys*, Vol. 24, No. 3, September 1992, pp. 219-291
- [45] John Canny, “The Complexity of Robot Motion Planning”. Cambridge, MA: MIT press, 1998
- [46] Jing Ren and Kenneth A. McIsaac, “A Hybrid-Systems Approach to Potential Field Navigation for a Multi-Robot Team”, *IEEE Proceedings of 2003, International Conference on Robotics & Automation*, Taipei, Taiwan, September 14-19, 2003
- [47] Johannes J. Schneider and Scott Kirkpatrick, “Stochastic Optimization”, *Springer, Springer-Verlag Berlin Heidelberg* 2006
- [48] Howie Choset and Keiji Nagatani, “Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization without Explicit Localization”, *IEEE Transaction on Robotics and Automation*, Vol.17, No.2, April 2001, pp. 125-137.

Appendix: MATLAB Simulation Code

```
%===== Information ===== Obstacle==Attractor==Start==Goal===== %

% Simulator for Potential Field based motion planning -- 2D
% Using Generalized Gaussian models for attractors and repulsors
%===== %

clear all
close all

mov = avifile('Temp.avi')

numAIn = 1; % # of attractors
numRIn = 4; % # of repulsors

%Positions of attractors and repulsors
paramA = [50 70 50]; %===== (ax, ay, variance)

%=== Sparsely spaced obstacles ===== %

paramR = [48 35 5 5; 50 55 5 5; 55 40 2.5 3; 56.5 60 2.5 3; 1 1 1 1; 1 1 1 1; 1 1 1
1]; %==(rx, ry, variance ,C)

%=== Different scenarios ===== %
%=== Different location for repulsor ===== %
%=== Closely spaced obstacles ===== %

% k = 1;
% x = 20;
% for y = 0:3:30
%     paramR(k,:) = [x y 2 1];
%     k = k + 1;
% end
% for x = 22:3:30
%     paramR(k,:) = [x y 2 1];
%     k = k + 1;
% end
% for x = 40:3:56
%     paramR(k,:) = [x y 2 1];
%     k = k + 1;
% end
% for y = 32:3:70
%     paramR(k,:) = [x y 2 1];
%     k = k + 1;
```

```

% end
% for x = 55:3:70
%   paramR(k,:) = [x y 2 1];
%   k = k + 1;
% end
% m = 32;
% for n = 0:3:16
%   paramR(k,:) = [m n 2 1];
%   k = k + 1;
% end
% for m = 34:3:50
%   paramR(k,:) = [m n 2 1];
%   k = k + 1;
% end
% for m = 60:3:70
%   paramR(k,:) = [m n 2 1];
%   k = k + 1;
% end
% for n = 18:3:70
%   paramR(k,:) = [m n 2 1];
%   k = k + 1;
% end
% x = 30;
% for y = 30:3:50
%   paramR(k,:) = [x y 2 1];
%   k = k + 1;
% end
% o = 40;
% for p = 30:3:50
%   paramR(k,:) = [o p 2 1];
%   k = k + 1;
% end
% o = 50;
% for p = 0:3:15
%   paramR(k,:) = [o p 2 1];
%   k = k + 1;
% end
% o = 60;
% for p = 0:3:15
%   paramR(k,:) = [o p 2 1];
%   k = k + 1;
% end
% numRIn = k-1;
%=====

```

```

target_location = [1 1];
target_location(1) = paramA(1,1);
target_location(2) = paramA(1,2);

% Link parameters
%=====
linkLen = 5;
linkLen12 = 5;
linkLen13 = 5;
varHead = 50;
%=====
flag = 0;
tp0 = 1;
flag_two = 0;
stop1 = 0;
stop2 = 0;
km = 0.3; %angular velocity coefficient

%Simulation parameters: initial conditions
%=====
NSTEP = 600;
stepSize = 0.2; %===== Speed (rate)
x0 = [50 20]; %===== Start point
theta0 = pi*(-105/180.0);
theta012 = pi*(-120/180.0);
theta013 = pi*(-105/180.0);
%=====
x1 = x0;

theta1 = theta0;
theta12 = theta012;
theta13 = theta013;

%===== Position of links with respect to head =====
xHead = x1;
xTail = xHead + [linkLen*cos(theta1), linkLen*sin(theta1)];
xTail12 = xTail + [linkLen12*cos(theta12), linkLen12*sin(theta12)];
xTail13 = xTail12 + [linkLen13*cos(theta13), linkLen13*sin(theta13)];

xtg(1,:) = x1;
xtgT(1,:) = xTail;
xtgT12(1,:) = xTail12;
xtgT13(1,:) = xTail13;

xtgTheta(1) = theta1;
xtgTheta12(1) = theta12;

```



```

xtgTheta13(1) = theta13;
flagr=0;

for k = 1:NSTEP
    z = k;

    % ===== for more links use loop
    xHead = x1;
    xTail = xHead + [linkLen*cos(theta1), linkLen*sin(theta1)];
    xTail12 = xTail + [linkLen12*cos(theta12), linkLen12*sin(theta12)];

    %=====for one link separation==== for more than one location or link use
loop=====
    if x1(1) < 57 && x1(1) > 53
        if x1(2) < 52 && x1(2) > 48
            flagr = 1;
        end
    end

    if flagr == 0
        xTail13 = xTail12 + [linkLen13*cos(theta13), linkLen13*sin(theta13)];
    end

    %====Find Next Location and heading direction=====%
    if stop1 == 0

        [theta1, theta12, theta13, x1] = Next_location(x1, theta1, theta12, theta13, paramA,
paramR, numRIn);

        xtgTheta(k+1) = theta1;
        xtgTheta12(k+1) = theta12;
        xtgTheta13(k+1) = theta13;

        xtg(k+1,:) = x1;
        xtgT(k+1,:) = x1 + [linkLen*cos(theta1), linkLen*sin(theta1)];
        xtgT12(k+1,:) = xtgT(k+1,:) + [linkLen12*cos(theta12), linkLen12*sin(theta12)];

        if flagr == 0
            xtgT13(k+1,:) = xtgT12(k+1,:) + [linkLen13*cos(theta13), linkLen13*sin(theta13)];
        end

    %====To find whether or not it is stuck.. at local minima=====%

    xtg3 = xtg(k+1,:);
    xtg2 = xtg(k,:);
    tp1 = xtg2(1);

```

```

tp2 = xtg3(1);
tp3 = xtg2(2);
tp4 = xtg3(2);

tp5(tp0) = tp2 - tp1;
tp6(tp0) = tp4 - tp3;
tp0 = tp0 + 1;

%=====target reached??!!=====
if abs(target_location(1) - tp1) < 0.1
    if abs(target_location(2) - tp3) < 0.1
        flag = 5;
    end
end

if tp0 == 11
    tp0 = 1;
    [flag_two] = local_minima(flag_two,tp5,tp6);
end

if flag_two == 2
    flag_two = 0;
    [x1, theta1, theta12, theta13] = Simulated_annealing(numAIn, numRIn, xtg(k-1,:),
paramA, paramR, theta1, theta12, theta13);
end

end

figure(1)
xTrans = 0;
xmin=0;
xmax=100;
ymin=0;
ymax=100;
dx = 1;
dy = 1;
[xx,yy] = meshgrid(xmin:dx:xmax,ymin:dy:ymax);

V3 = V3Generate_field(numAIn, numRIn, xx,yy,paramA,paramR);

%===== To Increase speed of simulation =====
if k > 250
    if k < 300
        hold on
        contourf(xx + xTrans,yy,V3,1);%colormap
        colormap cool
    end
end

```

```

        grid
    end
end
hold off
%=====

plot(xtg(:,1)+xTrans,xtg(:,2),'r','linewidth',2)
hold on

%====show the position of robots at each step====%
if stop1 == 0
text(xtg(k,1), xtg(k,2), sprintf('%s%d','R',k),'FontSize',15)
end
hold on
plot([xTail(1),xHead(1)], [xTail(2),xHead(2)], 'bl','linewidth',5)
grid
plot([xTail12(1),xTail(1)], [xTail12(2),xTail(2)], 'bl','linewidth',5)
grid

if flagr == 0
    plot([xTail13(1),xTail12(1)], [xTail13(2),xTail12(2)], 'bl', 'linewidth',5)
    grid
end
if flagr == 1
    plot([xTail13(1),(xTail13(1)+1)], [xTail13(2),(xTail13(2)+1)], 'bl','linewidth',5)
    grid
end

for variable_x = 1:numAIn
    text(paramA(variable_x,1), paramA(variable_x,2), '@')
end
text(60,38,'R')

hold on
contour(xx + xTrans,yy,V3,1,'g','linewidth',2);%colormap
%colormap cool %====(to change color map, contour plots)====%
grid

hold on
plot(xtgT(:,1) + xTrans,xtgT(:,2),'g','linewidth',1)
grid
hold off
hold on
plot(xtgT12(:,1) + xTrans,xtgT12(:,2),'g','linewidth',1)
grid
hold off

```

```

hold on
plot(xtgT13(:,1) + xTrans,xtgT13(:,2),'g','linewidth',1)
grid
hold off

axis([20 70 0 70])

Mt = getframe;

if flag == 5
flag = 0;
stop1 = 1;
end
end

mov = close(mov)

%===== Next_location =====%
%===== Function : Next Location =====%

function [theta1,theta12,theta13,x1] = Next_location(x1, theta1In, theta12In, theta13In,
paramAIn, paramRIn, numRIn1)

delta = 1e-12;
theta1 = theta1In;
theta12 = theta12In;
theta13 = theta13In;
paramR = paramRIn;
stepSize = 0.2;
linkLen = 5;
linkLen12 = 5;
linkLen13 = 5;
varHead = 50;

xHead = x1;
xTail = xHead + [linkLen*cos(theta1), linkLen*sin(theta1)];
xTail12 = xTail + [linkLen12*cos(theta12), linkLen12*sin(theta12)];
xTail13 = xTail12 + [linkLen13*cos(theta13), linkLen13*sin(theta13)];

numAIn = 1; % # of attractors
numRIn = numRIn1; % # of repulsors

paramA = paramAIn;

```

```

%===== f: function, g: gradient, H: hessian =====%

[f,g,H] = fnPF(numAIn, numRIn, x1,paramA,paramR);

%=====Newton's Method : Hessian Matrix =====%
epsilon = max( 0, delta - 0.5*( H(1,1)+H(2,2) - sqrt((H(1,1)-H(2,2))^2+4*H(1,2)^2) ) );
dtm = (epsilon+H(1,1))*(epsilon+H(2,2)) - H(1,2)^2;
B(1,1) = (epsilon+H(2,2))/dtm;
B(2,2) = (epsilon+H(1,1))/dtm;
B(1,2) = -H(1,2)/dtm;
B(2,1) = B(1,2);

%===== for control law =====%
dk = - (B*g)';

paramATail = [xHead(1),xHead(2),varHead];
paramATail12 = [xTail(1),xTail(2),varHead];
paramATail13 = [xTail12(1),xTail12(2),varHead];

numAInTail = 1;
numAInTail12 = 1;
numAInTail13 = 1;

%===== for tails =====%

[fTail,gTail,HTail] = fnPF(numAInTail, numRIn, xTail,paramATail,paramR);

epsilon = max( 0, delta - 0.5*( HTail(1,1)+HTail(2,2) - sqrt((HTail(1,1)-
HTail(2,2))^2+4*HTail(1,2)^2) ) );
dtm = (epsilon+HTail(1,1))*(epsilon+HTail(2,2)) - HTail(1,2)^2;
BT(1,1) = (epsilon+HTail(2,2))/dtm;
BT(2,2) = (epsilon+HTail(1,1))/dtm;
BT(1,2) = -HTail(1,2)/dtm;
BT(2,1) = BT(1,2);

[fTail12,gTail12,HTail12] = fnPF(numAInTail12, numRIn, xTail12, paramATail12,
paramR);

epsilon = max( 0, delta - 0.5*( HTail12(1,1)+HTail12(2,2) - sqrt((HTail12(1,1)-
HTail12(2,2))^2+4*HTail12(1,2)^2) ) );
dtm = (epsilon+HTail12(1,1))*(epsilon+HTail12(2,2)) - HTail12(1,2)^2;
BT2(1,1) = (epsilon+HTail12(2,2))/dtm;
BT2(2,2) = (epsilon+HTail12(1,1))/dtm;
BT2(1,2) = -HTail12(1,2)/dtm;

```

```
BT2(2,1) = BT2(1,2);
```

```
[fTail13,gTail13,HTail13] = fnPF(numAInTail13, numRIn, xTail13, paramATail13,  
paramR);
```

```
epsilon = max( 0, delta - 0.5*( HTail13(1,1)+HTail13(2,2) - sqrt((HTail13(1,1)-  
HTail13(2,2))^2+4*HTail13(1,2)^2) ) );
```

```
dtm = (epsilon+HTail13(1,1))*(epsilon+HTail13(2,2)) - HTail13(1,2)^2;
```

```
BT3(1,1) = (epsilon+HTail13(2,2))/dtm;
```

```
BT3(2,2) = (epsilon+HTail13(1,1))/dtm;
```

```
BT3(1,2) = -HTail13(1,2)/dtm;
```

```
BT3(2,1) = BT3(1,2);
```

```
% Orientation as the reference direction
```

```
refDirFlag = 1;
```

```
if refDirFlag == 1
```

```
    refVect = [cos(theta1), sin(theta1)];
```

```
    coordAngle = pi/2;
```

```
end
```

```
% Negative gradient of head as the reference direction
```

```
if refDirFlag ~= 1
```

```
    %refVect = -g;
```

```
    refVect = - (BT*gTail)';
```

```
    coordAngle = -pi/2;
```

```
end
```

```
refRot = [cos(coordAngle), -sin(coordAngle); sin(coordAngle), cos(coordAngle)];
```

```
refVect2 = refRot*refVect';
```

```
inProd = -gTail(1)*refVect2(1) - gTail(2)*refVect2(2);
```

```
if inProd >= 0
```

```
    omegaDir = 1;
```

```
else
```

```
    omegaDir = -1;
```

```
end
```

```
%dk= -g; %Gradient
```

```
% Orientation as the reference direction link 2
```

```
refDirFlag12 = 1;
```

```
if refDirFlag12 == 1
```

```
    refVect12 = [cos(theta12), sin(theta12)];
```

```
    coordAngle12 = pi/2;
```

```
end
```

```

% Negative gradient of head as the reference direction
if refDirFlag12 ~= 1
    %refVect12 = -g;
    refVect12 = - (BT2*gTail12)';
    coordAngle12 = -pi/2;
end

refRot12 = [cos(coordAngle12), -sin(coordAngle12); sin(coordAngle12),
cos(coordAngle12)];
refVect122 = refRot12*refVect12';

inProd12 = -gTail12(1)*refVect122(1) - gTail12(2)*refVect122(2);
if inProd12 >= 0
    omegaDir12 = 1;
else
    omegaDir12 = -1;
end
%dk= -g; %Gradient

% Orientation as the reference direction link 3

refDirFlag13 = 1;
if refDirFlag13 == 1
    refVect13 = [cos(theta13), sin(theta13)];
    coordAngle13 = pi/2;
end

% Negative gradient of head as the reference direction
if refDirFlag13 ~= 1
    %refVect13 = -g;
    refVect13 = - (BT3*gTail13)';
    coordAngle13 = -pi/2;
end

refRot13 = [cos(coordAngle13), -sin(coordAngle13); sin(coordAngle13),
cos(coordAngle13)];
refVect123 = refRot13*refVect13';

inProd13 = -gTail13(1)*refVect123(1) - gTail13(2)*refVect123(2);
if inProd13 >= 0
    omegaDir13 = 1;
else
    omegaDir13 = -1;
end
% dk= -g; %Gradient

```

```

x2(1) = x1(1) + stepSize*dk(1)/sqrt(dk(1)^2+dk(2)^2);
x2(2) = x1(2) + stepSize*dk(2)/sqrt(dk(1)^2+dk(2)^2);

theta2 = theta1 + 0.05*omegaDir;
theta122 = theta12 + 0.05*omegaDir12;
theta123 = theta13 + 0.05*omegaDir13;

x10 = x1;
x1 = x2;
theta1 = theta2;
theta12 = theta122;
theta13 = theta123;

return

%===== fnPF : find gradient and hessian =====%

% Calculate function/gradient/Hessian info at position xin
% xin = [x,y]
% paramAin = [ax1,ay1,var1; ax2,ay2,var2; ...]
% paramRin = [rx1,ry1,var1,C1; rx2,ry2,var2,C2; ...]
% [f,g,H] = [function, gradient, Hessian]
%
% Attractor: function [f,g,H]=fnAttractor(xin,paramA)
% Repulsor: function [f,g,H]=fnRepulsor(xin,paramR)

function [f,g,H]=fnPF(numAIn, numRIn, xin, paramAIn, paramRIn)
    numA = numAIn;
    numR = numRIn;

    f = 0;
    g = [0 0];
    H = [0 0; 0 0];

    % =====Attractors=====
    for k = 1:numA
        paramA = paramAIn(k,:);
        [fa,ga,Ha]=fnAttractor(xin,paramA);
        f = f + fa;

        if nargout > 1 % Gradient evaluated at x,y
            g = g + ga;
        end
    end

```



```

        if nargout > 2 % Hessian evaluated at x,y
            H = H + Ha;
        end
    end

    % =====Repulsors=====
    for k = 1:numR
        paramR = paramRIn(k,:);
        [fr,gr,Hr]=fnRepulsor(xin,paramR);
        f = f + fr;

        if nargout > 1 % Gradient evaluated at x,y
            g = g + gr;
        end
        if nargout > 2 % Hessian evaluated at x,y
            H = H + Hr;
        end
    end

    g = g*1000;
    return

%==== function : Attractor =====
%=====
% Calculate gradient information at position xin
% xin = [x,y]
% param = [ax,ay,var]
% [f,g,H] = [function, gradient, Hessian]
%=====

function [f,g,H]=fnAttractor(xin,param)
    x = xin(1);
    y = xin(2);
    ax = param(1);
    ay = param(2);
    var = param(3);

    Amp = 300;
    % Compute the objective function value at x,y

    f = 1 - exp(-((x-ax)^2 + (y-ay)^2)/(2*var^2));
    f1 = f - 1 ;

    % Compute the gradient at x,y

```

```

    if nargout > 1 % fun called with two output arguments

% Gradient of the function evaluated at x,y
    g(1) = f1*(ax-x)/(var^2);
    g(2) = f1*(ay-y)/(var^2);
end

% Compute the Hessian at x,y

if nargout > 2
    % Hessian evaluated at x,y
    H(1,1) = f1*((ax-x)^2 - var^2)/(var^4);
    H(2,2) = f1*((ay-y)^2 - var^2)/(var^4);
    H(1,2) = f1*(ax-x)*(ay-y)/(var^4);
    H(2,1) = H(1,2);
end

return

%===== function : Repulsor =====%
%=====
% Calculate gradient information at position xin
%  xin  = [x,y]
%  param = [rx,ry,var,C]
%  [f,g,H] = [function, gradient, Hessian]
%=====

function [f,g,H]=fnRepulsor(xin,param)
    x = xin(1);
    y = xin(2);
    rx = param(1);
    ry = param(2);
    var = param(3);
    C = param(4);

% Compute the objective function value at x,y

    Axy = ((x-rx)^2 + (y-ry)^2)/(var^2);
    f = exp(-0.5*Axy^C);

% Compute the gradient at x,y

```

```

if nargout > 1 % fun called with two output arguments
    % Gradient of the function evaluated at x,y
    g(1) = f*Axy^(C-1)*C*(rx-x)/(var^2);
    g(2) = f*Axy^(C-1)*C*(ry-y)/(var^2);
end

% Compute the Hessian at x,y

if nargout > 2
    % Hessian evaluated at x,y
    H(1,1) = f*( Axy^(2*C-2)*(C*(rx-x)/(var^2))^2 - Axy^(C-1)*C/(var^2) - Axy^(C-
2)*2*C*(C-1)*(rx-x)^2/(var^4) );
    H(2,2) = f*( Axy^(2*C-2)*(C*(ry-y)/(var^2))^2 - Axy^(C-1)*C/(var^2) - Axy^(C-
2)*2*C*(C-1)*(ry-y)^2/(var^4) );
    H(1,2) = f*C*(rx-x)*(ry-y)/(var^4)*( C*Axy^(2*C-2) - 2*(C-1)*Axy^(C-2) );
    H(2,1) = H(1,2);
end

return

%===== Local Minima =====%
%===== Whether or not its stuck condition =====%

function [flag_two] = local_minima(flag_twoIn,tp5In,tp6In)

    tp5 = tp5In;
    tp6 = tp6In;
    flag_two = flag_twoIn;
    tp55 = 0;
    tp66 = 0;
    for m = 1:10
        tp55 = tp5(m) + tp55;
        tp66 = tp6(m) + tp66;
    end
    tp55 = tp55 / 10;
    tp66 = tp66 / 10;

    tp55=abs(tp55);
    tp66=abs(tp66);

    if tp55 < 0.05
        if tp66 < 0.05
            flag_two = flag_two + 1;
        end
    end

```

```

end

return

%===== Simulated Annealing =====%
%===== Local Minima Recovery =====%

function [x1,theta1,theta12,theta13] = Simulated_annealing(numAIn, numRIn, xtg,
paramA, paramR, theta1, theta12, theta13)

    no_solution = 0;
    no_solution1 = 0;
    no_solution2 = 0;

%=== find navigation function for neighbor =====%

    [f_minima] = fnPF_nbr(numAIn, numRIn, xtg,paramA,paramR);
    temp_th = 0.1430;
    temp = 20;
    x1=xtg;
    t=1;
    while temp > temp_th
        temp = temp * 0.9;

%=== find bunch of neighbors =====%

        [x_random] = neighbours(x1);

%===== Avoid back tracking =====%

        x20(t,:)=x1;
        t = t+1;
        f_best = 1;

        for S = 1:9
            [f_random(S,:)] = fnPF_nbr(numAIn, numRIn, x_random(S,:), paramA, paramR);

            flag_three = 0;
            for t1 = 1:(t-1)
                if x20(t1,:) == x_random(S,:)
                    flag_three = 1;
                end
            end
        end
    end

```

```

        if (flag_three == 1)
            continue
        else
            if f_random(S,:) <= f_best
                f_best = f_random(S,:);
                R = S;
            end
        end
    end
end

%===== finding the delta E , to check  $0.2 \leq e^{(-\Delta E / T)}$  =====%

delta_f = f_best - f_minima;
f_temp = delta_f / temp;
x_random(R,:)

if delta_f < 0
    x21 = x1;
    x1 = x_random(R,:);
    hold off

%=== Move towards the local target ===%
[theta1,theta12,theta13] = local_goal(x1, x21, theta1, theta12, theta13, paramR);
no_solution = 1;

    break
elseif (0 <= delta_f) && (delta_f <= 0.99)
    if (f_temp <= 1.60) && (f_best < 1)
        x21 = x1;
        x1 = x_random(R,:);
        hold off

%=== Move towards the local target ===%
[theta1,theta12,theta13] = local_goal(x1, x21, theta1, theta12, theta13, paramR);
    else
        no_solution1 = 1;

        break
    end
else
    no_solution2 = 1;
    break
end
end
end

```

```
return
```

```
%===== Function for Neighbor =====%
```

```
%===== Find value of navigation function for neighbors =====%
```

```
function [f_nbr]=fnPF_nbr(numAIn, numRIn, xin,paramAIn,paramRIn)
```

```
    numA = numAIn;
```

```
    numR = numRIn;
```

```
    f_nbr = 0;
```

```
%===== Attractors =====%
```

```
for k = 1:numA
```

```
    paramA = paramAIn(k,:);
```

```
    [fa_nbr]=fnAttractor_nbr(xin,paramA);
```

```
    f_nbr = f_nbr + fa_nbr;
```

```
end
```

```
%===== Repulsors =====%
```

```
for k = 1:numR
```

```
    paramR = paramRIn(k,:);
```

```
    [fr_nbr]=fnRepulsor_nbr(xin,paramR);
```

```
    f_nbr = f_nbr + fr_nbr;
```

```
end
```

```
return
```

```
%===== Function Attractor =====%
```

```
%===== Attractor Function for neighbor =====%
```

```
function [f]=fnAttractor_nbr(xin,param)
```

```
    x = xin(1);
```

```
    y = xin(2);
```

```
    ax = param(1);
```

```
    ay = param(2);
```

```
    var = param(3);
```

```
%===== Compute the objective function value at x,y =====%
```

```
    f = 1 - exp(-((x-ax)^2 + (y-ay)^2)/(2*var^2));
```

```
return
```

```
%===== Function Repulsor =====%
%===== Repulsor Function for neighbor =====%
```

```
function [f]=fnRepulsor_nbr(xin,param)
```

```
    x = xin(1);
    y = xin(2);
    rx = param(1);
    ry = param(2);
    var = param(3);
    C = param(4);
```

```
%=== Compute the objective function value at x,y ===%
```

```
    Axy = ((x-rx)^2 + (y-ry)^2)/(var^2);
    f = exp(-0.5*Axy^C);
```

```
return
```

```
%===== Neighbor =====%
```

```
%===== Find Neighbors of current location =====%
```

```
function [x_random] = neighbours (x1)
```

```
    x_1 = x1(:,1);
    y_1 = x1(:,2);
    o = 1;
```

```
    for m = (x_1 - 0.5): 0.5 : (x_1 + 0.5)
        for n = (y_1 - 0.5): 0.5 : (y_1 + 0.5)
            x_random(o,:) = [m,n];
            o = o + 1;
        end
    end
```

```
%===== To find randomize bunch of neighbor, use matlab function randi() with the loop =====%
```

```
return
```

```

%===== Local Goal : SA =====%
%===== Move towards local goal set by annealing =====%

function [theta1,theta12,theta13] = local_goal(local_goal, local_start, local_theta,
local_theta12, local_theta13, paramRIn)

numAIn = 1; % # of attractors
numRIn = 4; % # of repulsors

%Positions of attractors and repulsors : attractor is local goal

paramA = [local_goal(1) local_goal(2) 50; 50 100 100]; %(ax,ay,var)

paramR = paramRIn;
paramA2 = [50 70 50];

% Link parameters
linkLen = 5;
linkLen12 = 5;
linkLen13 = 5;
varHead = 50;

flag = 0;
flag_two = 0;
km = 0.3; %angular velocity coefficient

%Simulation parameters: initial conditions
NSTEP2 = 8;
stepSize = 0.1; %Speed (rate)
x0 = local_start; %Start point

x1 = x0;
theta1 = local_theta;
theta12 = local_theta12;
theta13 = local_theta13;
xHead = x1;
xTail = xHead + [linkLen*cos(theta1), linkLen*sin(theta1)];
xTail12 = xTail + [linkLen12*cos(theta12), linkLen12*sin(theta12)];
xTail13 = xTail12 + [linkLen13*cos(theta13), linkLen13*sin(theta13)];

xtg(1,:) = x1;
xtgT(1,:) = xTail;
xtgT12(1,:) = xTail12;
xtgT13(1,:) = xTail13;

```



```

xtgTheta(1) = theta1;
xtgTheta12(1)=theta12;
xtgTheta13(1)=theta13;

for k2 = 1:NSTEP2

    xHead = x1;
    xTail = xHead + [linkLen*cos(theta1), linkLen*sin(theta1)];
    xTail12 = xTail + [linkLen12*cos(theta12), linkLen12*sin(theta12)];
    xTail13 = xTail12 + [linkLen13*cos(theta13), linkLen13*sin(theta13)];

    [f,g,H] = fnPF(numAIn, numRIn, x1,paramA,paramR);

    paramATail = [xHead(1),xHead(2),varHead];
    numAInTail = 1;
    [fTail,gTail,HTail] = fnPF(numAInTail, numRIn, xTail, paramATail, paramR);

    paramATail12 = [xTail(1),xTail(2),varHead];
    numAInTail12 = 1;
    [fTail12,gTail12,HTail12] = fnPF(numAInTail12, numRIn, xTail12, paramATail12,
paramR);

    paramATail13 = [xTail12(1),xTail12(2),varHead];
    numAInTail13 = 1;
    [fTail13,gTail13,HTail13] = fnPF(numAInTail13, numRIn, xTail13, paramATail13,
paramR);

    % Orientation as the reference direction
    refDirFlag = 1;
    if refDirFlag == 1
        refVect = [cos(theta1), sin(theta1)];
        coordAngle = pi/2;
    end

    % Negative gradient of head as the reference direction
    if refDirFlag ~= 1
        refVect = -g;
        coordAngle = -pi/2;
    end

    refRot = [cos(coordAngle), -sin(coordAngle); sin(coordAngle), cos(coordAngle)];
    refVect2 = refRot*refVect';

    inProd = -gTail(1)*refVect2(1) - gTail(2)*refVect2(2);
    if inProd >= 0
        omegaDir = 1;

```

```

else
    omegaDir = -1;
end
dk= -g;  %Gradient

% Orientation as the reference direction Link 2
refDirFlag12 = 1;
if refDirFlag12 == 1
    refVect12 = [cos(theta12), sin(theta12)];
    coordAngle12 = pi/2;
end

% Negative gradient of head as the reference direction
if refDirFlag12 ~= 1
    refVect12 = -g;
    coordAngle12 = -pi/2;
end

refRot12 = [cos(coordAngle12), -sin(coordAngle12); sin(coordAngle12),
cos(coordAngle12)];
refVect122 = refRot12*refVect12';

inProd12 = -gTail12(1)*refVect122(1) - gTail12(2)*refVect122(2);
if inProd12 >= 0
    omegaDir12 = 1;
else
    omegaDir12 = -1;
end
dk= -g;  %Gradient

% Orientation as the reference direction Link 3
refDirFlag13 = 1;
if refDirFlag13 == 1
    refVect13 = [cos(theta13), sin(theta13)];
    coordAngle13 = pi/2;
end

% Negative gradient of head as the reference direction
if refDirFlag13 ~= 1
    refVect13 = -g;
    coordAngle13 = -pi/2;
end

refRot13 = [cos(coordAngle13), -sin(coordAngle13); sin(coordAngle13),
cos(coordAngle13)];
refVect123 = refRot13*refVect13';

```

```

inProd13 = -gTail13(1)*refVect123(1) - gTail13(2)*refVect123(2);
if inProd13 >= 0
    omegaDir13 = 1;
else
    omegaDir13 = -1;
end
dk= -g;  %Gradient

x2(1) = x1(1) + stepSize*dk(1)/sqrt(dk(1)^2+dk(2)^2);
x2(2) = x1(2) + stepSize*dk(2)/sqrt(dk(1)^2+dk(2)^2);

theta2 = theta1 + 0.05*omegaDir;
theta122 = theta12 + 0.05*omegaDir12;
theta123 = theta13 + 0.05*omegaDir13;
x10 = x1;
x1 = x2;

theta1 = theta2;
theta12 = theta122;
theta13 = theta123;
xtgTheta(k2+1) = theta1;
xtgTheta12(k2+1) = theta12;
xtgTheta13(k2+1) = theta13;

xtg(k2+1,:) = x1;
xtgT(k2+1,:) = x1 + [linkLen*cos(theta1), linkLen*sin(theta1)];
xtgT12(k2+1,:) = xtgT(k2+1,:) + [linkLen12*cos(theta12), linkLen12*sin(theta12)];
xtgT13(k2+1,:) = xtgT12(k2+1,:) + [linkLen13*cos(theta13), linkLen13*sin(theta13)];

figure(1)
xTrans = 0;

xmin=0;
xmax=100;
ymin=0;
ymax=100;
dx = 1;
dy = 1;

[xx,yy] = meshgrid(xmin:dx:xmax,ymin:dy:ymax);

V3 = V3Generate_field(numAIn, numRIn, xx,yy,paramA2,paramR);

```

```

hold on
contourf(xx + xTrans,yy,V3,1);%colormap
colormap cool
grid

%hold off
plot(xtg(:,1)+xTrans,xtg(:,2),'r','linewidth',2)
hold on
%show the position of robots at each step
text(xtg(k2,1), xtg(k2,2), sprintf('%s%d','R',k2),'FontSize',15)
hold on
plot([xTail(1),xHead(1)], [xTail(2),xHead(2)], 'bl','linewidth',5)
grid
plot([xTail12(1),xTail(1)], [xTail12(2),xTail(2)], 'bl','linewidth',5)
grid
plot([xTail13(1),xTail12(1)], [xTail13(2),xTail12(2)], 'bl','linewidth',5)
grid

for variable_x = 1:numAIn
    text(paramA2(variable_x,1), paramA2(variable_x,2), '@')
end

hold on
plot(xtgT(:,1) + xTrans,xtgT(:,2),'g','linewidth',1)
grid
hold off
hold on
plot(xtgT12(:,1) + xTrans,xtgT12(:,2),'g','linewidth',1)
grid
hold off
hold on
plot(xtgT13(:,1) + xTrans,xtgT13(:,2),'g','linewidth',1)
grid
hold off

axis([20 70 0 70])
Mt = getframe;

end
return

```

```

%===== V3Generate_field =====%
% Generate 3D data at grids (xx,yy)
% xx,yy: matrices of grid points
% paramAIn = [ax1,ay1,var1; ax2,ay2,var2; ...]
% paramRIn = [rx1,ry1,var1,C1; rx2,ry2,var2,C2; ...]
% Attractor: function V3=fnAContour(xin,paramA)
% Repulsor: function V3=fnRContour(xin,paramR)
%===== %

function V3 = V3Generate_2(numAIn,numRIn, xx,yy,paramAIn,paramRIn)
    numA = numAIn;
    numR = numRIn;

    V3 = 0;
    for k = 1:numA
        paramA = paramAIn(k,:);
        fa = fnAContour(xx,yy,paramA);

        V3 = V3 + fa;
    end

    for k = 1:numR
        paramR = paramRIn(k,:);
        fr = fnRContour(xx,yy,paramR);

        V3 = V3 + fr;
    end
    return

%===== function : Attractor : Contour =====%
%===== Create contours plots for attractive field =====%

function V3=fnAContour(xxIn,yyIn,paramA)
    xx = xxIn;
    yy = yyIn;
    ax = paramA(1);
    ay = paramA(2);
    var = paramA(3);

    % Compute the objective function value at x,y
    V3 = 1 - exp(-((xx-ax).^2 + (yy-ay).^2)/(2*var^2));
    return

```

```

%===== function : Repulsor : Contour =====%
%===== Create contours plots for repulsive field =====%

function V3=fnRContour(xxIn,yyIn,paramR)
    xx = xxIn;
    yy = yyIn;
    rx = paramR(1);
    ry = paramR(2);
    var = paramR(3);
    C = paramR(4);

    % Compute the objective function values at grids (xx,yy)
    Axy = ((xx-rx).^2 + (yy-ry).^2)/(var^2);
    V3 = exp(-0.5*Axy.^C);
return

```